Incremental Task and Motion Planning: A Constraint-Based Approach

Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki
Department of Computer Science, Rice University, Houston, TX 77005, USA; {ntd,zkk1,swarat,kavraki}@rice.edu

***Abstract*—We present a new algorithm for task and motion planning (TMP) and discuss the requirements and abstractions necessary to obtain robust solutions for TMP in general. Our *Iteratively Deepened Task and Motion Planning* (IDTMP) method is probabilistically-complete and offers improved performance and generality compared to a similar, state-of-the-art, probabilistically-complete planner. The key idea of IDTMP is to leverage incremental constraint solving to efficiently add and remove constraints on motion feasibility at the task level. We validate IDTMP on a physical manipulator and evaluate scalability on scenarios with many objects and long plans, showing order-of-magnitude gains compared to the benchmark planner and a four-times self-comparison speedup from our extensions. Finally, in addition to describing a new method for TMP and its implementation on a physical robot, we also put forward requirements and abstractions for the development of similar planners in the future.**

I. Introduction

Robots in the physical world must couple high-level decisions and geometric reasoning. The robot must plan over a *task-motion space*, combining discrete decisions about objects and actions with continuous decisions about collision-free paths. Efficient algorithms exist to solve each of these parts in isolation; however, integrating task and motion planning (TMP) presents algorithmic challenges both in scalability and completeness. Isolated task planning typically produces a single plan, whereas TMP may require alternate task plans based on motion level exploration. Isolated motion planning typically assumes a fixed configuration space during planning, whereas TMP may change the configuration space by moving objects. A key challenge for TMP is that proving the *nonexistence* of motion plans is difficult and unsolved for the general case [35, 60]. We directly consider alternate task plans, changing configuration spaces, and motion feasibility to introduce a new, probabilistically complete TMP algorithm.

We present a probabilistically-complete algorithm for TMP that offers improved performance and generality over the prior work for probabilistically-complete TMP in manipulation. We first discuss a set of reusable insights on TMP that underlie our approach (see Sec. II and Sec. IV). Our TMP algorithm extends constraint[1]-based task planning using the incremental solution capabilities of Satisfiability Modulo Theories (SMT) solvers to dynamically incorporate motion feasibility at the task level (see Sec. V). We additionally analyze domain assumptions on connectivity and algorithmic extensions that improve performance of our method while preserving probabilistic completeness (see Sec. VI). Finally, we validate our planner on a physical manipulator and show that it scales better with the number of objects and length of plan compared to the similar TMP method of [36] (see Sec. VII).

[1]*Constraint* is an unfortunately overloaded term. Our usage of "constraint" in this paper corresponds to the logical assertions of task planners based on satisfiability checking.

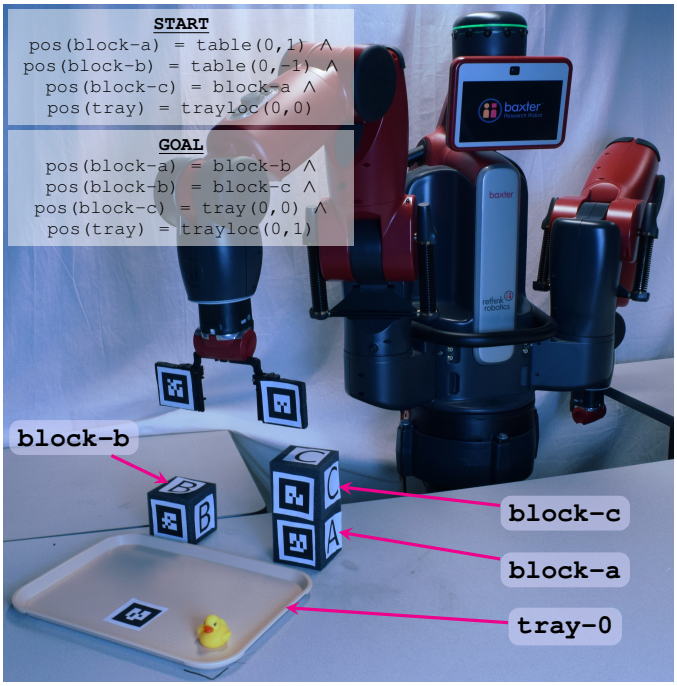


Fig. 1: Basic TMP Example. The robot must stack the blocks on the tray in a particular order, then move the tray. Planning must consider different actions to manipulate the different kinds of objects (blocks and tray), determine the order of these actions, and identify collision-free paths for the motion.

II. Challenges and Requirements

TMP combines continuous *motion* decisions about paths with discrete *task* decisions about objects and actions. The typical algorithms for independent task planning [37, 39, 43, 67] and motion planning [44, 53] are fundamentally different. Consequently, most TMP methods [9, 36, 69, 72] perform task planning and motion planning as separate, possibly interleaved, phases. We isolate and discuss the specific requirements for task planning, motion planning, and their interface in order to perform efficient and robust TMP.

A. Task Planning Requirements

Task planning finds a discrete sequence of actions to transition from a given start state to a desired goal condition [27].

The key requirement for the task planning phase of TMP is support for generating alternate plans. As we iterate between task planning and motion planning phases, feedback from the motion planner ideally influences the task planner to favor operators with previously computed motion plans or disfavor more difficult or potentially infeasible operators. The task planning layer must therefore be able to compute alternate plans for a domain and ideally reuse work from previous planning rounds to improve performance.

Secondarily, the task planner must support a sufficiently expressive specification format to model the desired domain. Previous work applied a variety of notations to the modeling of discrete robot tasks: temporal logics [3, 36, 47], structured and natural language [46, 59], logical knowledge bases [25, 75], the Planning Domain Definition Language (PDDL) [30, 61, 72], and context-free grammars [14, 58, 79]. Each notation provides advantages for certain domains or proprieties, e.g., safety properties are easily specified with temporal logics, action effects with PDDL, and hierarchies with grammars. Therefore, a general robotics task and motion planner would ideally be independent of the particular domain specification syntax, enabling the use of the most suitable notation.

B. Motion Planning Requirements

Geometric motion planning finds a collision-free path from a given start position to a desired goal [10].

Rearranging objects changes the the robot's configuration space, i.e., the valid joint positions. Moreover, some objects are kinematically coupled (e.g., blocks on the tray in Fig. 1). Thus, the motion planner must use underlying representations that permit both efficient updates and model object interactions.

C. Task-Motion Interface Requirements

TMP combines the discrete action selection of task planning with the continuous path generation of motion planning.

The primary requirement of the task-motion layer is to establish a correspondence between task operators and motion planning problems. Given the geometric scene, what is the corresponding task description? Given a task action or plan, what are the corresponding motion planning problems? The task-motion interface must translate between the low-level scene geometry and the high-level task descriptions.

A secondary requirement of the task-motion layer arises when we wish to ensure some form of *completeness* in planning. For the current state of the art in high-dimensional motion planning, probabilistic completeness is the best we can hope to achieve. A fundamental challenge for general, high-dimensional systems is that we cannot prove the non-existence of a motion plan [35, 60]. Consequently, we cannot definitively rule out an attempted task action because a motion planner was unable to find a concrete path in the allotted time; the motion planner may have failed because such a path indeed does not exist or merely because insufficient time was allotted to motion planning. Thus, to ensure probabilistically complete TMP, we must not eliminate failed task actions but instead set them aside to be later reattempted.

III. Related Work

A. Task Planning

Task planning is a well-established field, largely evolving from the pioneering work on STRIPS [27]. The currently dominant approaches for efficient task planning [76] are heuristic search [37, 39, 78] and constraint-based methods [43, 67, 68]. Logic programming is also used [55, 75].

We adopt the constraint-based task planning approach to leverage ongoing advances in solvers for *Satisfiability Modulo Theories* (SMT) [2, 16]. Typical constraint-based planners use boolean satisfiability formulas [43, 67]. SMT extends boolean satisfiability with rules (*theories*) for domains such as linear arithmetic. Compared to traditional SAT solvers, SMT solvers provide a more expressive and high-level interface with useful features for expressing constraints in robotics domains [62, 80]. Crucially, some SMT solvers [1, 15] enable *incremental* solving – adding and deleting constraints at run-time to produce alternate solutions. Our approach applies incremental solving to update constraints about motion feasibility.

B. Motion Planning

The major approaches for high-dimensional motion planning are heuristic search [32] and sampling-based [44, 53] methods. Heuristic search planners are sometimes used for manipulation [11], but it is challenging to find general heuristics that work for different manipulators. In contrast, sampling-based planners efficiently handle high degree-of-freedom (DOF) systems without robot-specific modifications, typically through constructing either multi-query roadmaps [44] or single-query trees [53]. However, sampling-based planners can offer only *probabilistic completeness*. Consequently, the failure of a sampling-based planner to find a plan does *not* prove such a plan does not exist. This challenge is a fundamental consideration in the design of our algorithm.

In this work, we do not modify the operation of the motion planner, but rather employ sampling-based planners from the Open Motion Planning Library [13]. Specifically, we use bidirectional Rapidly-exploring Random Trees (RRTs) [49]. However, our algorithm and its properties depend only on the probabilistic completeness of the underlying motion planner, so other choices for a motion planner are possible.

C. Task and Motion Planning

Most prior work on TMP focuses on performance over completeness or generality. [22] interleaves task and motion planning at the level of individual task actions, calling the motion planner directly from the task planner for feasibility checks using *semantic attachments*. [25] produces a knowledge base for household robots in the logic programming paradigm [56]. [51] applies geometric constraints to limit the motion planning space or prove motion infeasibility in special cases. Hierarchical Planning in the Now (HPN) [40, 41] interleaves planning and execution, reducing search depth but requiring

reversible actions when backtracking. Several related methods [17, 18, 19, 30] extend Hierarchical Task Networks (HTN) [26] with geometric primitives, using shared literals to control backtracking between the task and motion layer. [72] interfaces off-the-shelf task planners with an optimization-based motion planner [70] using a heuristic to remove potentially-interfering objects. [57] formulates the *motion* side of TMP as a constraint satisfaction problem over a discretized, preprocessed subset of the configuration space. [62, 80] use an SMT solver to generate task and motion plans from a static roadmap, employing plan outlines to guide the planning process in a manner similar to Hierarchical Task Networks [26]. FFRob [29] develops an FF-like [39] task-layer heuristic based on a lazily-expanded roadmap. Overall, these methods set aside the general challenge of ensuring probabilistic completeness of the overall approach. In contrast, we directly address the the challenge of probabilistically complete TMP.

Other works perform task and motion planning for differential or hybrid systems using sampling-based planners [42, 64, 65]. Differential dynamics, though necessary for some domains, poses challenges for completeness even in isolated motion planning [50]. In contrast, we consider purely geometric motion which is adequate for many manipulation tasks and for which there exist many probabilistically-complete motion planners [52, 13].

A smaller number of other task and motion planners also achieve probabilistic completeness. The aSyMov planner [8, 9, 31] combines a heuristic-search task planner based on metric-FF [38] with lazily-expanded roadmaps. Our proposed algorithm operates differently at all levels, yielding different performance characteristics from aSyMov. For example, aSyMov's composed roadmaps could be amortized over multiple runs but composing roadmaps for object interactions may be expensive, as acknowledged by the authors in [9]. In contrast, we motion plan anew each run, but efficiently update scene data structures to handle object interaction. The Synergistic Framework [65] and related methods [4, 5, 36] are similar to our proposed approach, but there are important differences in the underlying algorithms that suggest these methods may be complementary. The Synergistic Framework finds task plans through forward search, while we use constraint-based methods to efficiently generate task plans. There is also a distinction in the feedback between task and motion planners. Our approach incorporates geometric information from failed motion planning attempts via incremental constraint updates, while the Synergistic Framework uses feedback between the task and motion planner to guide a weighted forward-search which may assist difficult motion planning domains. [36] also focuses on the manipulation domain; in comparison, our method provides both more flexible abstractions than [36]'s domain-specific task-state graph and significant performance improvements for the benchmarks in Sec. VII. Thus, we present our new approach as a complementary alternative to prior work on probabilistically complete TMP.

Several related methods consider motion planning with movable objects. We view such works as special cases of TMP with restricted task actions. Navigation among movable obstacles (NAMO) [21, 54, 73, 74, 77], Minimum Constraint Removal/Displacement (MCR/MCD) [34, 35] find motion plans in the presence of movable obstacles. Rearrangement Planning bases the goal not on the robot's position but rather on positions of the objects to be moved [48]. In contrast to NAMO, MCR, MCD, and Rearrangement Planning, general TMP considers multiple, arbitrary task actions.

IV. Task-Motion Abstractions and Definition

We now formalize the abstractions for our approach to TMP. To formally define the TMP problem, we first separately define the task domain (see Def. 1) and motion domain (see Def. 3), then combine them in the task and motion domain (see Def. 5).

A. Task Domain

Task domains are typically defined in terms of states and actions using a variety of notations [25, 36, 72]. To support various notations, we define the notation-independent task domain as the following *formal language*:

Definition 1 (Task Language): The task language is a set of strings of actions, defined by $\mathfrak{L} = \big(\mathcal{P}, \mathcal{A}, \mathcal{E}, s^{[0]}, \mathcal{G}\big)$, where,

- $\mathcal{P}$ is the state space ranging over variables $p_0, \ldots, p_n$,
- $\mathcal{A}$ is the set of task operators, i.e., terminal symbols,
- $\mathcal{E} \subseteq (\mathbb{P}(\mathcal{P}) \times \mathcal{A} \times \mathbb{P}(\mathcal{P}))$ is the set of *symbolic transitions*. Each $e_i \in \mathcal{E}$ denotes transitions $\mathrm{pre}(a_i) \xrightarrow{a_i} \mathrm{eff}(a_i)$, where $\mathrm{pre}(a_i) \subseteq \mathcal{P}$ is the precondition set, $a_i \in \mathcal{A}$ is the operator, and $\mathrm{eff}(a_i) \subseteq \mathcal{P}$ is the effect set. We represent a concrete transition on a_i at step k from state $s^{[k]}$ to $s^{[k+1]}$ as $s^{[k+1]} = a_i(s^{[k]})$, where $s^{[k]} \in \mathrm{pre}(a_i)$, $s^{[k+1]} \in \mathrm{eff}(a_i)$, and for all state variables p_j independent of (i.e., free in) $\mathrm{eff}(a_i)$, ${p_j}^{[k+1]} = {p_j}^{[k]}$,
- $s^{[0]} \in \mathcal{P}$ is the start state,
- $\mathcal{G} \subseteq \mathcal{P}$ is the set of accept states, i.e., the task goal.

Definition 2 (Task Plan): A *task plan* $\mathbf{A}$ is a string in the task language $\mathfrak{L}$, i.e., $\mathbf{A} \in \mathfrak{L}$, where $\mathbf{A} = \big(a^{[0]}, a^{[1]}, \ldots, a^{[h]}\big)$, $a^{[k]} \in \mathcal{A}$, $s^{[k]} \in \mathrm{pre}\big(a^{[k]}\big)$, $s^{[k+1]} = a^{[k]}(s^{[k]})$, and $s^{[h]} \in \mathcal{G}$.

Note that Def. 1 and Def. 2 are not a new task notation but an abstraction for many existing notations, e.g., [23, 61].

B. Motion Domain

Robot manipulators are modeled as kinematic trees or scene graphs of joints and links [33]; these structures underlie popular software packages such as OpenRave [20], KDL [71], MoveIt! [12], Gazebo [45], and ROS [28, 81]. In contrast to prior frameworks, TMP problems require (1) translation between the motion and task domains and (2) efficiently changing the kinematic topology of the scene graph as objects are grasped, moved, and released. To support TMP requirements for task-motion translation and efficient, dynamic updates, we modify and streamline typical scene graph representations such as [81] as follows:

Definition 3 (Scene Graph): $\gamma = (\mathcal{Q}, \mathcal{L}, \mathcal{F})$, where

- $\mathcal{Q} \subseteq \mathbb{R}^n$ is a space of configurations,
- $\mathcal{L}$ is a finite set of unique frame labels,
- $\mathcal{F}$ is a finite set of kinematic frames (graph nodes), such that each frame $f_\ell = (\ell, \varrho_\ell, \varsigma_\ell, \mu_\ell)$, where,

- $\ell \in \mathcal{L}$ is the unique label of frame of f_ℓ
- $\varrho_\ell \in \mathcal{L}$ is the label of the parent of frame of f_ℓ, indicating graph edge connections
- $\varsigma_\ell : \mathcal{Q} \mapsto \mathcal{SE}(3)$, maps from the configuration space to the workspace pose of f_ℓ relative to its parent ϱ_ℓ, indicating graph edge values
- μ_ℓ is a rigid body mesh representing geometry attached to f_ℓ.

The global or absolute $\mathcal{SE}(3)$ transform of any frame ℓ in the scene graph is the product of relative transforms from root 0 to frame ℓ: ${}^0\mathcal{S}_i * \ldots * {}^j\mathcal{S}_\ell$, where ${}^a\mathcal{S}_b$ is the transform in $\mathcal{SE}(3)$ from parent a to child b and ${}^{\varrho_b}\mathcal{S}_b = \varsigma_b(q)$. Typically, it is not possible to explicitly represent the configuration space $\mathcal{Q}$ of high DOF manipulators, so sampling-based planners use specialized collision checkers [63] to determine whether a given configuration is valid based on the absolute $\mathcal{SE}(3)$ poses ${}^0\mathcal{S}_\ell$ of all geometry μ_ℓ.

Common TMP operations such as grasping an object change the collision-free configuration space, which we represent by changing the topology of the scene graph. For example, grasping `block-c` in Fig. 1 changes its parent label from its support object, i.e., `block-a`, to the gripper label. We formalize such topological changes to the scene graph using a general *reparent* operation. Reparenting changes the parent label of frame ℓ to ϱ'_ℓ and computes the new relative pose ${}^{\varrho'_\ell}\mathcal{S}_\ell$ that preserves ${}^0\mathcal{S}_\ell$ by ${}^{\varrho'_\ell}\mathcal{S}_\ell = ({}^0\mathcal{S}_{\varrho'_\ell})^{-1} * ({}^0\mathcal{S}_\ell)$.

Definition 4 (Motion Plan): A *motion plan* is a sequence of neighboring configurations $\mathbf{Q} = \left(q^{[0]}, q^{[1]}, \ldots, q^{[n]}\right)$ such that each $q^{[k]} \in \mathcal{Q}$ and $\left\|q^{[i+1]} - q^{[i]}\right\| < \epsilon_q$, for some small ϵ_q. The initial configuration is $\text{first}(\mathbf{Q}) = q^{[0]}$, and the final configuration is $\text{last}(\mathbf{Q}) = q^{[n]}$.

C. Task and Motion Domain

We define the domain $\mathfrak{D}$ for TMP problems using our task language and scene graph. A key detail is the translation between task and motion state. We *abstract* a scene graph to a single task state where the task-state position $s \in \mathcal{P}$ is given by the parent ϱ_ℓ of each object ℓ in the scene graph. Task actions such as grasping or placing objects correspond to a set of configurations due to multiple grasp positions or redundancy of the manipulator. We *refine* the effect of a task operator to a single scene graph and multiple configurations. We define $\mathfrak{D}$ as follows:

Definition 5 (Task-Motion Domain):
$\mathfrak{D} = \left(\mathfrak{L}, \sigma^{[0]}, \lambda_\alpha, \lambda_\rho, \Omega\right)$

- $\mathfrak{L}$ is the task language, where $\mathcal{P} = \mathcal{P}_m \times \mathcal{P}_t$ represents the motion component $\mathcal{P}_m$ and non-motion component $\mathcal{P}_t$ of task state,
- $\sigma^{[0]} = \left(s^{[0]}, \gamma^{[0]}, q^{[0]}\right)$ is initial the task-motion state: task state $s^{[0]}$, scene graph $\gamma^{[0]}$ and configuration $q^{[0]}$,
- $\lambda_\alpha : \Gamma \mapsto \mathcal{P}_m$ abstracts the scene graph $\gamma \in \Gamma$ to the motion component task state $s_m \in \mathcal{P}_m$,
- $\lambda_\rho : \Gamma \times \mathcal{A} \mapsto \mathbb{P}(\mathcal{Q}) \times \Gamma$ refines the initial scene graph $\gamma^{[k]} \in \Gamma$ and task operator $a^{[k]} \in \mathcal{A}$ to a motion planning goal (a set of configurations) $\Theta^{[k]} \subseteq \mathcal{Q}$ for the action and a final scene graph $\gamma^{[k+1]} \in \Gamma$ via reparenting frames,

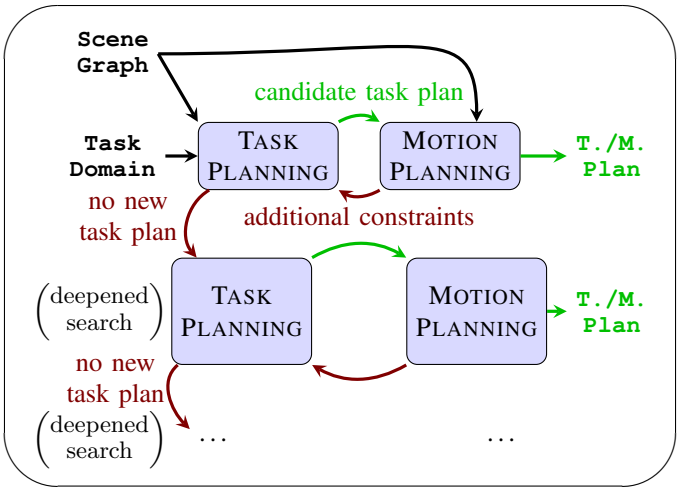


Fig. 2: Diagram of IDTMP. We incrementally incorporate motion feasibility information into the task planner via incremental constraint solving. The bound increases [43] of our constraint-based task planner are coupled to timeout increases for sampling-based motion planning.

- $\Omega \subseteq \Gamma \times \mathcal{P}$ is the goal condition.

Definition 6 (Task and Motion Plan): A *task and motion plan* is a sequence of task operators and motion plans, $\mathbf{T} = \left(\left(a^{[0]}, \mathbf{Q}^{[0]}\right), \ldots, \left(a^{[h]}, \mathbf{Q}^{[h]}\right)\right)$ where $\left(a^{[0]}, \ldots, a^{[h]}\right) \in \mathfrak{L}$ and for each step k, $\left(\Theta^{[k]}, \gamma^{[k+1]}\right) = \lambda_\rho(a^{[k]}, \gamma^{[k]})$ such that $\text{last}\left(\mathbf{Q}^{[k]}\right) \in \Theta^{[k]} \wedge \text{first}\left(\mathbf{Q}^{[k+1]}\right) = \text{last}\left(\mathbf{Q}^{[k]}\right)$.

V. IDTMP

We present a new algorithm for task and motion planning (TMP), Iteratively Deepened Task and Motion Planning (IDTMP), Algorithm 1, that uses a constraint-based task planner to generate candidate task plans (see line 9), then uses a sampling based motion planner to check plan feasibility (see line 17). Fig. 2 graphically depicts the operation. Constraint-based planners encode the task planning problem over a bounded *step horizon* as a logical formula and compute a solution using a constraint solver, iteratively increasing the step horizon (see line 14) [43] until a plan is found. Sampling-based motion planners terminate either when a motion plan is found or when a *sampling horizon*, or timeout, is exceeded (see line 19). We couple a progressively increasing *sampling horizon* of the motion planner to the increasing *step horizon* of the task planner (see line 14). We use an SMT solver for constraint-solving. Crucially, we use the SMT solver's constraint stack to efficiently generate alternate task plans over the increasing step horizon, then if necessary later `pop` (see line 12) the additional constraints (see line 21) to *revisit* task plans with increased motion search horizons, ensuring that the motion planner has sufficient time to identify feasible paths.

A. Task Planning Implementation

We develop a custom task planner by extending the constraint-based planning method [43, 52, 66]. Typical task planners are optimized for single-shot queries whereas for TMP, it may be necessary to generate alternate task plans. Our

task planner efficiently generates alternate plans by leveraging modern, *incremental* SMT solvers. Incremental SMT solvers maintain a stack of constraints or assertions and can efficiently perform repeated satisfiability checks as constraints are pushed onto and popped from the constraint stack [1, 15]. Our use of incremental SMT solving to update constraints is a new feature compared to previous use of SMT solvers in TMP [62, 80].

1) Background on Constraint-Based Planning: Constraint-based planners encode the planning domain as a logical formula, then use a constraint solver – typically, a boolean satisfiability solver – to find a satisfying variable assignment for that formula, corresponding to the plan [52, p69]. The formula contains variables for the task state and the action to take for a fixed number of steps h. Given a domain with state variables $p_0, \ldots, p_m$ and actions $a_0, \ldots, a_n$, the set of formula variables for h steps is $\left\{{p_i}^{[k]} \mid i \in 0 \ldots m, k \in 0 \ldots h\right\} \cup \left\{{a_j}^{[k]} \mid j \in 0 \ldots n, k \in 0 \ldots h\right\}$. The formula itself asserts the transitions from Def. 1 and Def. 2. Specifically, the start state holds a step 0 (see line 3) and goal condition holds at step h (see line 5 and line 16), and states and actions obey the transitions $\mathcal{E}$ as follows (see line 3 and line 13):

- A selected action implies its preconditions and effects (i.e., the $\mathrm{pre}(a_i)$ and $\mathrm{eff}(a_i)$ in $\mathcal{E}$): for every action a_i and step k, ${a_i}^{[k]} \implies \left(\mathrm{pre}(a_i)^{[k]} \wedge \mathrm{eff}(a_i)^{[k+1]}\right)$.
- State remains the same unless changed by an action's effect: for every state variable p_i and step k, $\left({p_i}^{[k]} = {p_i}^{[k+1]}\right) \vee \left({a_j}^{[k]} \vee \ldots \vee {a_\ell}^{[k]}\right)$, where $a_j, \ldots, a_\ell$ are the actions that modify p_i.
- Only one action is taken at a time: for every action a_i and step k, ${a_i}^{[k]} \implies \left(\neg {a_0}^{[k]} \wedge \ldots \wedge \neg {a_{i-1}}^{[k]} \wedge \neg {a_{i+1}}^{[k]} \wedge \ldots \wedge \neg {a_n}^{[k]}\right)$.

The planner progressively increases step count h until the formula is satisfiable, indicating a valid plan where the action at each step k is given by which variables ${a_i}^{[k]}$ are true in the satisfying assignment.

2) An Incremental Task Planner: We extend previous work on constraint-based task planning by using an incremental SMT solver for constraint satisfaction. In Algorithm 1, the key, novel feature of our task planner compared to typical constraint-based planners is the ability to efficiently add (line 21) and remove (line 12) constraints based on motion feasibility to generate and revisit alternate task plans (line 10). The incremental SMT solver maintains constraints using a stack of *scopes*, where each scope is a container for a set of constraints. The planner pushes scopes onto the stack, adds new constraints to the top scope on the stack, and later pops the top scope from the stack thus removing the constraints within that scope [2]. For our planner, the top scope holds constraints for the current step horizon, e.g., the goal condition is true at the end (see line 16).

Algorithm 1: IDTMP

Input: $\left(\mathfrak{L}, \sigma^{[0]}, \lambda_\alpha, \lambda_\rho, \Omega\right)$: Task-Motion Domain
Output: $\mathbf{T}$: Task-Motion Plan

1 $(s^{[0]}, \gamma^{[0]}, q^{[0]}) \leftarrow \sigma^{[0]}$; /* Start State */
2 $(h, t) \leftarrow (1, t_0)$; /* Initial Horizons */
/* ϕ: formula for task domain */
3 $\phi \leftarrow s^{[0]} \wedge \lambda_\alpha(\gamma_0)^{[0]} \wedge$ (transitions of $\mathfrak{L}$ at step 0) ;
4 push(ϕ); /* Push scope */
5 $\phi \leftarrow \phi \wedge \left(\lambda_\alpha\left(\Omega_\Gamma\right)\right)^{[h]} \wedge {\Omega_\mathcal{P}}^{[h]}$; /* Goal at h */
6 $\mathbf{T} \leftarrow \emptyset$; /* $\mathbf{T}$: Task-Motion Plan */
7 **while** $\emptyset = \mathbf{T}$ **do**
8 $\quad \mathbf{A} \leftarrow \emptyset$; /* A: Task Plan */
9 $\quad$ **while** $\emptyset = \mathbf{A}$ **do** /* Task Planning */
10 $\quad\quad \mathbf{A} \leftarrow$ Incremental_SMT(ϕ);
11 $\quad\quad$ **if** $\emptyset = \mathbf{A}$ **then**
12 $\quad\quad\quad$ pop(ϕ); /* Pop scope */
13 $\quad\quad\quad \phi \leftarrow \phi \wedge$ (transitions of $\mathfrak{L}$ at step h) ;
14 $\quad\quad\quad (h, t) \leftarrow (h + 1,\ t + \Delta t)$;
15 $\quad\quad\quad$ push(ϕ); /* Push scope */
16 $\quad\quad\quad \phi \leftarrow \phi \wedge \left(\lambda_\alpha(\Omega_\Gamma)\right)^{[h]} \wedge {\Omega_\mathcal{P}}^{[h]}$; /* Goal */
17 $\quad$ **foreach** $a^{[k]} \in \mathbf{A}$ **do** /* Motion Refine */
18 $\quad\quad (\Theta^{[k]}, \gamma^{[k+1]}) \leftarrow \lambda_\rho(\gamma^{[k]}, a^{[k]})$; /* Goal */
19 $\quad\quad \mathbf{Q}^{[k]} \leftarrow$ motion_plan$(\gamma^{[k]}, q^{[k]}, \Theta^{[k]}, t)$;
20 $\quad\quad$ **if** $\emptyset = \mathbf{Q}^{[k]}$ **then** /* Motion Failed */
21 $\quad\quad\quad \phi \leftarrow \phi \wedge$ (New Constraints) ;
22 $\quad\quad\quad \mathbf{T} \leftarrow \emptyset$;
23 $\quad\quad\quad$ **break;**
24 $\quad\quad$ **else**
25 $\quad\quad\quad q^{[k+1]} \leftarrow \mathrm{last}(\mathbf{Q}^{[k]})$;
26 $\quad\quad\quad \mathbf{T} \leftarrow \mathrm{append}(\mathbf{T}, (a^{[k]}, \mathbf{Q}^{[k]}))$;
27 **return** $\mathbf{T}$;

B. Motion Planning Implementation

We use a sampling-based, single-query motion planner – specifically RRT-Connect [49, 13] – to instantiate task actions. Using a probabilistically complete, single-query planner backed by our scene graphs (see Def. 3) allows us to ensure probabilistic completeness (see subsection V-D) over changing configuration spaces but at the cost of repeated motion planning computation, which we partially address through extensions in Sec. VI. Though roadmap based planners such as [6, 7] offer the potential for efficient repeated queries, the changing configuration spaces of TMP pose challenges for their use.

For each step k (see line 17), we determine the goal based on operator $a^{[k]}$ and the current scene graph $\gamma^{[k]}$ (see line 18), then attempt to find a motion plan (see line 19). If the motion planner cannot find a plan for $a^{[k]}$ within the current sampling horizon t (see line 20), we give additional constraints to the task planner to select a different operator (see line 21).

C. Task-Motion Interaction

The task-motion interface connects the task layer and motion layer through domain-specific functions λ_α and λ_ρ (see Def. 5). Abstraction function λ_α translates scene graphs to task state (see line 3, line 5, and line 16). Refinement function λ_ρ translates task operators to motion planning problems

(see line 18). For the specific domain in our experiments (see Fig. 1), λ_α produces a task state indicating the object placements, i.e., the parent frame label ϱ_ℓ of blocks and trays, and λ_ρ produces motion planning goals to transfer a block, stack a block, or push a tray.

We use the incremental feature of SMT solvers to efficiently generate alternate task plans which the motion planner attempts to refine. If the motion planner fails to refine a task plan, we add additional task constraints to produce the alternate plan (line 21). In the simple case, we could constrain the task planner to enumerate plans; a more efficient extension is presented to subsection VI-B. To enumerate plans, the additional constraint asserts that action variable assignments ${a_i}^{[k]}$ be different from that of the previously generated plan:

$$\neg \bigwedge_{k=0}^{h} \left(\bigwedge_{i=0}^{n} \left({a_i}^{[k]\prime} = {a_i}^{[k]} \right) \right) \tag{1}$$

Then, the next satisfiability check (line 10) will produce a different task plan $\mathbf{A}$ if one exists.

Since failure of the motion planner to refine operator a_i does not prove a_i is impossible, we later reconsider a_i with a greater motion planning horizon. While retrying motion planning duplicates computation, it is necessary for probabilistic completeness; we mitigate the extra computation through extensions in Sec. VI. We reconsider operators using the SMT solver's constraint stack. When our planner cannot find a different task plan at the current *step horizon*, we (a) pop the top constraint scope, removing the constraints it contains (see line 12), (b) increment both task planning *step horizon* h the motion planning *sampling horizon* t and (see line 14), and finally (c) push a new scope for future constraints (see line 15). Increasing the step horizon results in new task plans of longer length, even if the same constraints are later re-added. Thus we search for longer task plans through the incremented step horizon, and we search longer for motion plans through the incremented sampling horizon, notably reconsidering the previously failed task operators.

D. Completeness

We now prove the completeness properties of our basic algorithm.

Theorem 1: Naïve IDTMP is probabilistically complete.

Proof Outline: The constraint-based task planner is complete [43], so it enumerates all task plans as we increment the step horizon. The bidirectional RRT motion planner is probabilistically complete [49], so given sufficient time, the probablility of finding a plan, if one exists, approaches one. For each increment of task-planning step count, we revisit all previous task plans of lower count and attempt motion refinement with greater planning time. Because we iteratively increase the motion planning time for all potential task plans, the probability of successfully refining a valid task plan, if one exists, approaches one as time increases. □

VI. Completeness-Preserving Extensions

We now extend the initial IDTMP presented in Sec. V to improve performance while maintaining probabilistic completeness for typical cases. We formally define a basic assumption on connectivity of the configuration space for our extensions to preserve probabilistic completeness.

A. Basic Connectivity Assumption

In the general case, different plans to move an object may create disconnected configuration-space regions, such as in Fig. 3. Naïve IDTMP handles this possibility by retrying motion planning operations – both those that previously failed and previously succeeded. Failed operations must be retried because in general we are unable to prove the nonexistence of motion plans. Successful operations must also be retried to ensure exploration of different, disconnected final configurations. However, by focusing on typical manipulation environments where final configurations are connected, we can extend our algorithm to improve performance. We define a connectedness assumption to extend IDTMP as follows:

Definition 7 (Connected Configuration Set): A configuration set Θ is *connected* if there is a motion plan between all configurations in Θ:

$$\mathcal{C}(\Theta) \triangleq (\forall q_i, q_j \in \Theta,\ (\exists \mathbf{Q},\ \mathrm{first}(\mathbf{Q}) = q_i \wedge \mathrm{last}(\mathbf{Q}) = q_j))$$

Now we define postcondition configuration sets for plans and task states. The connectedness of these configuration sets enables our extensions to preserve probabilistic completeness.

Definition 8 (Plan Post-Configuration): The plan post-configuration $\mathcal{E}_{\mathcal{AQ}}$ maps from task plan $\mathbf{A} \in \mathfrak{L}$ to the set of valid final configurations $\Theta \subseteq \mathcal{Q}$ for that task plan,

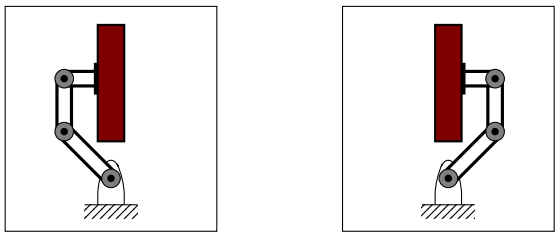
$$\mathcal{E}_{\mathcal{AQ}} : \mathfrak{L} \mapsto \mathbb{P}(\mathcal{Q}).$$

The post-configuration of a task operator to place an object is constrained by possible configurations that place the object in the desired destination. We can approximate the post-configuration by sampling valid placement configurations. Thus, plan post-configurations are determined by the final operator rather than the entire plan.

Definition 9 (State Post-Configuration): The state post-configuration $\mathcal{E}_{0\mathcal{Q}}$ maps from an initial state $\sigma^{[0]} \in \mathcal{P} \times \mathcal{Q}$ and a final task state $s^{[h]} \in \mathcal{P}$ to the set of valid final configurations $\Theta \subset \mathcal{Q}$ for the plans that reach $s^{[h]}$ from $\sigma^{[0]}$,

$$\mathcal{E}_{0\mathcal{Q}} : \mathcal{P} \times \mathcal{Q} \times \mathcal{P} \mapsto \mathbb{P}(\mathcal{Q}).$$

The state post-configuration $\mathcal{E}_{0\mathcal{Q}}$ is the union of plan post-configurations $\mathcal{E}_{\mathcal{AQ}}$ for plans satisfying the initial and final



(a) (b)

Fig. 3: Example object placements that cause disconnected post-configurations by "blocking-in" the robot. For this 3-DOF planar robot, the two sides of the workspace are disconnected.

condition of $\mathcal{E}_{0\mathcal{Q}}$:

$$\mathcal{E}_{0\mathcal{Q}}(\sigma^{[0]}, s^{[h]}) = \left\{ q \in \mathcal{E}_{\mathcal{AQ}}(\mathbf{A}) \,|\, \sigma^{[0]} \models \text{pre}(\mathbf{A}) \wedge s^{[h]} \models \text{eff}(\mathbf{A}) \right\} \quad (2)$$

Based on our connectivity assumption for these sets, we extend our algorithm and maintain probabilistic completeness.

B. Incremental Constraints

We reduce motion planning time spent attempting infeasible actions by generalizing motion planning failures. This failure generalization does not affect completeness when different task plans do not change the reachable configurations. If a state post-configuration is connected, i.e., $\mathcal{C}(\mathcal{E}_{0\mathcal{Q}}(\sigma^{[0]}, s)) = \top$, then the nonexistence of a motion plan from one configuration q_0 in $\mathcal{E}_{0\mathcal{Q}}(\sigma_0, s)$ to some configuration q_n implies that no plan to q_n exists for any configuration in $\mathcal{E}_{0\mathcal{Q}}(\sigma_0, s)$. Though we cannot in general prove nonexistence of motion plans, we instead heuristically defer retrying these failed motion plans at the current motion planning horizon. If the motion planner failed to refine operator a_i at state s, this may be because such a plan does not exist. Instead of constraining the task planner to produce any different plan as (1), we instead rule out, for the current horizon level, all plans that attempt a_i from state s for any step. The following constraint asserts that operator a_i not be attempted at state s for all steps:

$$\bigwedge_{k=0}^{h} \left(s^{[k]} \implies \neg {a_i}^{[k]} \right) \quad (3)$$

Then, when we deepen the step and sampling horizons, we likewise pop the added constraint of form (3) so that we will later reattempt these operators with greater search depth.

Using constraint (3), we generalize motion planning failures to all task steps. Such incremental constraint updates also directly apply to additional geometric information from the motion layer. For example, the specific objects preventing an action could be estimated using MCR [35] or the heuristic method of [72], further improving generalization of (3).

C. Completeness-preserving Plan Cache

We can reuse feasible motion plans without affecting completeness whenever a different plan to move an object does not change the reachable configurations. If a plan post-configuration is connected, i.e., $\mathcal{C}(\mathcal{E}_{\mathcal{AQ}}(\mathbf{A})) = \top$, then the reachable goal configurations are the same starting from all configurations in the plan post-configuration $\mathcal{E}_{\mathcal{AQ}}(\mathbf{A})$. Consequently, we need consider only a single motion plan for task plan $\mathbf{A}$ to capture all reachable configurations from $\mathcal{E}_{\mathcal{AQ}}(\mathbf{A})$. Conversely, if the post-configuration is not connected, i.e., $\mathcal{C}(\mathcal{E}_{\mathcal{AQ}}(\mathbf{A})) = \bot$, then reachable goal configurations will be different for the disconnected configuration regions. Thus, considering only a single motion plan neglects configurations reachable from the disconnected portion of $\mathcal{E}_{\mathcal{AQ}}(\mathbf{A})$.

We use plan post-configuration connectivity to implement a completeness preserving cache of motion plan prefixes. For a given task plan $\mathbf{A}$ with some prefix α, if $\mathcal{C}(\mathcal{E}_{\mathcal{AQ}}(\alpha)) = \top$, we first check for a cached motion plan for α, using it if it exists. Otherwise, we try to refine α, and if we find a motion plan, we add it to the cache. This eliminates naïve IDTMP's duplication of motion planning work for plan prefixes.

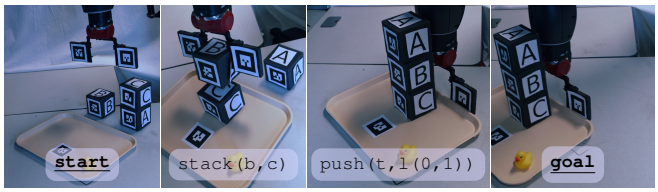


Fig. 4: Physical validation of IDTMP. The robot iteratively plans over actions for transferring blocks, stacking blocks, and pushing the tray and generating collision-free paths for the selected actions. The resulting task-motion plan is successfully executed on the physical robot.

VII. Experimental Results

We validate IDTMP on a physical robot and test scalability for simulated scenes. The physical validation demonstrates that IDTMP works for real planning problems with multiple types of actions. We compare the scalability of IDTMP against the planner in [36], which is the closest to our method in terms of philosophy and performance guarantees. Both methods similarly couple the task and motion planner, and both offer similar guarantees on probabilistic completeness. The simulated scalability tests demonstrate that IDTMP improves performance compared to the benchmark planner [36].

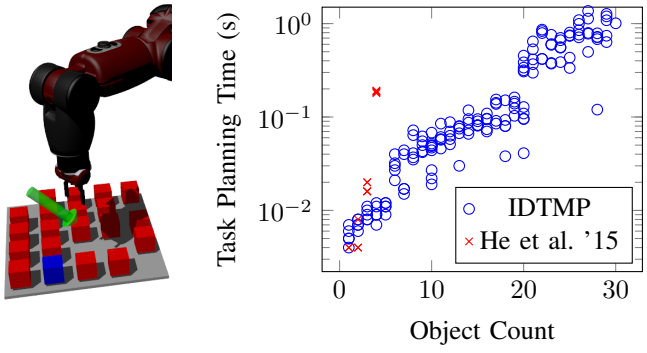
Our tests use simulated and physical Rethink Robotics Baxter manipulators. We use Z3 4.3.2 [15] as our backend SMT solver, the RRT-Connect [49] implementation in OMPL [13] for motion planning, FCL [63] for collision checking, and POV-Ray [24] for visualization. The benchmarks were conducted on on an Intel® i7-4790 under Linux 3.16.0-4. We randomly generate the benchmark scenes, resulting in small variance in the results. Though the test domains use axis-aligned grasps, this is not a limitation of our method, and arbitrary grasps, e.g., based on precomputed grasp quality metrics, are possible.

A. Physical Validation

We validate IDTMP on a physical manipulator for the scenario in Fig. 1, where the robot must stack the blocks on the tray and then push the tray. This domain demonstrates the object coupling of the scene graph (see Def. 3) during tray pushing, where for each object (i.e., block) frame ℓ, its parent ϱ_ℓ (i.e. other block or tray) is the other object on which it rests. Fig. 4 shows the robot executing the plan, demonstrating that our overall system works for physical scenarios.

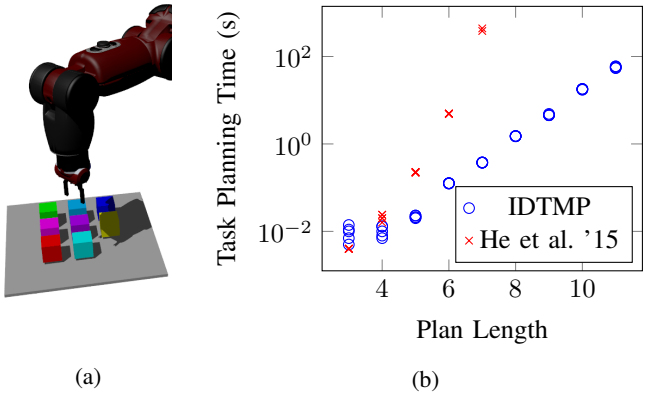
B. Scalability Tests

We first test scalability of IDTMP over increasing number of objects (see Fig. 5). While task planning in IDTMP does scale exponentially with the number of objects, it still performs task planning for tens of objects in around one second. In comparison, [36] scales worse than IDTMP for task planning with increasing object count. Above five objects, task planning



(a) (b)

Fig. 5: Scenario testing scalability as the object count, and correspondingly the discrete state space size, increase. (a) The robot must move the marked object (blue) to the center of the board (green arrow), removing the object that was there. (b) compares the task-planning performance of IDTMP and [36] over five trials for each object count. Motion-planning times are comparable between both methods.



(a) (b)

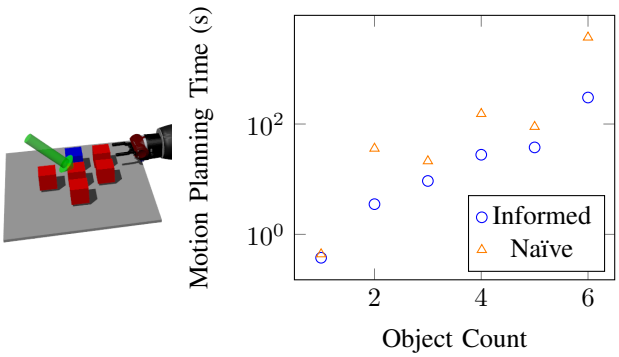
Fig. 6: Scenario testing scalability as necessary plan length increases. (a) The robot must move every block to a different goal location. (b) compares task-planning performance of IDTMP and [36] over five trials for each length. Motion planning times were comparable between both methods.

using [36] took several minutes. Motion planning performance for [36] and IDTMP are similar as both use a sampling-based motion planner in the same way to refine candidate task plans.

Next, we test the scalability of IDTMP for increasing length of the task plan that must be computed (see Fig. 6). Task planning time scales exponentially with increasing plan length, taking about 10 seconds to compute a plan that is 10 task actions long In comparison, [36] scales worse than IDTMP for task planning in the plan length test. Motion planning time for [36] is similar to IDTMP, just as in the object count test.

C. Benchmark of IDTMP Extensions

Finally, we test the benefit of our completneness-preserving extensions to IDTMP by rerunning the object count test (see Fig. 5a) but with the grasping pose constrained to one side of the object, resulting in many infeasible operations due



(a) (b)

Fig. 7: Comparison of Naïve and Informed IDTMP. (a) the object count test (see Fig. 5) was rerun with grasp poses constrained to one side of the object to cause infeasible actions. (b) compares naïve and informed IDTMP planning, averaged over five runs per object count. Our extensions improve motion planning performance by an average of 4.75 times. Task planning time is not significantly changed.

to blocking objects (see Fig. 7). Task planning times are unchanged, but the informed constraints speed up motion planning an average of 4.75 times by reducing reattempted motion planning for infeasible operations.

VIII. Conclusion

We have discussed the challenges and requirements of TMP and presented a new TMP algorithm. First, the TMP requirements and abstractions we present in Sec. II and Sec. IV underlie our planner. Moreover, these abstractions model the key features of task-motion domains and may aid the development and analysis of other task-motion planners. Second, our IDTMP algorithm is probabilistically complete, handles domains with various actions, and models kinematic coupling. Though the simple form of our algorithm duplicates work at the motion planning level in order to achieve probabilistic completeness over changing configuration spaces, we mitigate this issue through geometric connectivity requirements for typical domains, enabling algorithmic extensions to improve performance. We validate IDTMP on a physical Baxter manipulator and show that IDTMP provides improved scalability to object count and plan length compared to the manipulation framework of He et al. [36], a previously developed, similar task and motion planner. Finally, our incremental constraint update approach is general to many types of geometric information, and compatible constraints could be generated based on related methods [35, 51, 57, 72] to extend this work.

Acknowledgments

NTD, SC, and LEK are supported in part by NSF CCF-1514372, CCF-1162076, and IIS 1317849. ZKK is supported in part by an RUE associated with IIS 1317849. We thank Keliang He for his assistance in evaluating [36].

References

[1] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Int. Conf. on Computer-Aided Verification (CAV)*, pages 171–177. Springer, 2011.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. www.SMT-LIB.org.

[3] Calin Belta, Volkan Isler, and George J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *Trans. on Robotics (T-RO)*, 21 (5):864–874, 2005.

[4] Amit Bhatia, Lydia E. Kavraki, and Moshe Y. Vardi. Motion planning with hybrid dynamics and temporal goals. In *Conf. on Decision and Control (CDC)*, pages 1108–1115, Atlanta, GA, 2010. IEEE.

[5] Amit Bhatia, Matthew R. Maly, Lydia E. Kavraki, and Moshe Y. Vardi. Motion planning with complex goals. *Robotics & Automation Magazine (RAM)*, 18(3):55–64, Sept. 2011. ISSN 1070-9932.

[6] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy PRM. In *Int. Conf. on Robotics and Automation (ICRA)*, volume 1, pages 521–528. IEEE, 2000.

[7] Robert Bohlin and Lydia E. Kavraki. *A Randomized Algorithm for Robot Path Planning Based on Lazy Evaluation*, pages 221–249. Kluwer Academic Publishers, 2001.

[8] Stéphane Cambon, Fabien Gravot, and Rachid Alami. A robot task planner that merges symbolic and geometric reasoning. In *Eu. Conf. on Artificial Intelligence (ECAI)*, volume 16, page 895, 2004.

[9] Stéphane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *Int. Journal of Robotics Research (IJRR)*, 28(1):104–126, 2009.

[10] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[11] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Single and dual-arm motion planning with heuristic search. *Int. Journal of Robotics Research (IJRR)*, 3(2): 305–320, February 2014.

[12] Ioan A. Şucan and Sachin Chitta. MoveIt!, 2015. http://moveit.ros.org.

[13] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The open motion planning library. *Robotics & Automation Magazine (RAM)*, 19(4):72–82, 2012.

[14] Neil T. Dantam and Mike Stilman. The motion grammar: Analysis of a linguistic method for robot control. *Trans. on Robotics (T-RO)*, 29(3):704–718, 2013.

[15] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[16] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[17] Lavindra de Silva, Akhilesh Kumar Pandey, and Rachid Alami. An interface for interleaved symbolic-geometric planning and backtracking. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 232–239. IEEE, 2013.

[18] Lavindra de Silva, Amit Kumar Pandey, Mamoun Gharbi, and Rachid Alami. Towards combining HTN planning and geometric task planning. In *Robotics: Science and Systems Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*, 2013.

[19] Lavindra de Silva, Mamoun Gharbi, Akhilesh Kumar Pandey, and Rachid Alami. A new approach to combined symbolic-geometric backtracking in the context of human-robot interaction. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 3757–3763. IEEE, 2014.

[20] Rosen Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010. URL http://www.programmingvision.com/rosen_diankov_thesis.pdf.

[21] Mehmet Dogar and Siddhartha Srinivasa. A framework for push-grasping in clutter. In *Robotics: Science and Systems (RSS)*, 2011.

[22] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *Towards Service Robots for Everyday Environments*, pages 99–115. Springer, 2012.

[23] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2004.

[24] Alexander Enzmannn, Chris Young, and Lutz Kretzschmar. *Ray tracing worlds with POV-Ray*. Sams, 1994.

[25] Esra Erdem, Erdi Aker, and Volkan Patoglu. Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intelligent Service Robotics*, 5(4):275–291, 2012.

[26] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128, 1994.

[27] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1972.

[28] Tully Foote. tf: The transform library. In *Technologies for Practical Robot Applications (TePRA)*, pages 1–6. IEEE, 2013.

[29] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFRob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*, pages 179–195. Springer, 2015.

[30] Mamoun Gharbi, Raphael Lallement, and Rachid Alami.

Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 6360–6365. IEEE, 2015.

[31] Fabien Gravot, Stéphane Cambon, and Rachid Alami. aSyMov: a planner that deals with intricate symbolic and geometric problems. In *Int. Symp. on Robotics Research (ISRR)*, pages 100–110. Springer, 2005.

[32] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Trans. Systems Science and Cybernetics*, 4 (2):100–107, 1968.

[33] Richard Scheunemann Hartenberg and Jacques Denavit. *Kinematic synthesis of linkages*. McGraw-Hill, 1964.

[34] Kris Hauser. Minimum constraint displacement motion planning. In *Robotics: Science and Systems*, 2013.

[35] Kris Hauser. The minimum constraint removal problem with three robotics applications. *Int. Journal of Robotics Research (IJRR)*, page 0278364913507795, 2013.

[36] Keliang He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. Towards manipulation planning with temporal logic specifications. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 346–352, Seattle, WA, 2015. IEEE.

[37] Malte Helmert. The fast downward planning system. *Journal Artificial Intelligence Resesearch (JAIR)*, 26: 191–246, 2006.

[38] Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research (JAIR)*, pages 291–341, 2003.

[39] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, pages 253–302, 2001.

[40] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 1470–1477. IEEE, 2011.

[41] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Integrated task and motion planning in belief space. *Int. Journal of Robotics Research (IJRR)*, 32(9-10):1194–1227, August/September 2013.

[42] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In *American Control Conf. (ACC)*, pages 735–742. IEEE, 2012.

[43] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Int. Joint. Conf. on Artifial Intelligence (IJCAI)*, volume 99, pages 318–325, 1999.

[44] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Trans. on. Robotics and Automation (T-RO)*, 12(4):566–580, 1996.

[45] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. IEEE, 2004.

[46] Hadas Kress-Gazit, Georgios E Fainekos, and George J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[47] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *Trans. on Robotics (T-RO)*, 25(6):1370–1381, 2009.

[48] Athanasios Krontiris and Kostas E. Bekris. Dealing with difficult instances of object rearrangement. In *Robotics: Science and Systems (RSS)*, 2015.

[49] James J. Kuffner and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Int. Conf. on Robotics and Automation (ICRA)*, volume 2, pages 995–1001. IEEE, 2000.

[50] Tobias Kunz and Mike Stilman. Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete. In *Algorithmic Foundations of Robotics XI*, pages 233–244. Springer, 2015.

[51] Fabien Lagriffoul, Dimitar Dimitrov, Alessandro Saffiotti, and Lars Karlsson. Constraint propagation on interval bounds for dealing with geometric backtracking. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 957–964. IEEE, 2012.

[52] Steven M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[53] Steven M. LaValle and James J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In Bruce Donald, Kevin Lynch, and Daniela Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A. K. Peters/CRC Press, 2001.

[54] Martin Levihn, Jonathan Scholz, and Mike Stilman. Hierarchical decision theoretic planning for navigation among movable obstacles. In *Algorithmic Foundations of Robotics X*, pages 19–35. Springer, 2013.

[55] Vladimir Lifschitz. Answer set planning. In *Logic Programming and Nonmonotonic Reasoning*, pages 373–374. Springer, 1999.

[56] Vladimir Lifschitz. What is answer set programming?. In *AAAI*, volume 8, pages 1594–1597, 2008.

[57] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3684–3691. IEEE, 2014.

[58] Damian M. Lyons and Michael A. Arbib. A formal model of computation for sensory-based robotics. *Trans. on Robotics and Automation*, 5(3):280–293, 1989.

[59] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pages 403–415. Springer, 2013.

[60] Zoe McCarthy, Timothy Bretl, and Seth Hutchinson. Proving path non-existence using sampling and alpha

shapes. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 2563–2569. IEEE, 2012.

[61] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *PDDL – the planning domain definition language*. AIPS-98 Planning Competition Committee, 1998.

[62] Srinivas Nedunuri, Shashank Prabhu, Mark Moll, Swarat Chaudhuri, and Lydia E. Kavraki. SMT-based synthesis of integrated task and motion plans from plan outlines. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 655–662. IEEE, 2014.

[63] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 3859–3866. IEEE, 2012.

[64] Erion Plaku and Gregory D. Hager. Sampling-based motion and symbolic action planning with geometric and differential constraints. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 5002–5008. IEEE, 2010.

[65] Erion Plaku, Lydia E. Kavraki, and Moshe Y. Vardi. Motion planning with dynamics by a synergistic combination of layers of planning. *Trans. on Robotics (T-RO)*, 26(3):469 – 482, 2010 2010.

[66] Jussi Rintanen. Engineering efficient planners with SAT. In *Eu. Conf. on Artificial Intelligence (ECAI)*, pages 684–689, 2012.

[67] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[68] Jussi Rintanen. Madagascar: Scalable planning with sat. In *8th International Planning Competition (IPC-2014)*, pages 66–70, 2014.

[69] Peter Schüller, Volkan Patoglu, and Esra Erdem. Levels of integration between low-level reasoning and task planning. In *AAAI Workshop on Intelligent Robotic Systems*, 2013.

[70] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *Int. Journal of Robotics Research (IJRR)*, 33 (9):1251–1270, 2014.

[71] Ruben Smits, Herman Bruyninckx, and Erwin Aertbeliën. KDL: Kinematics and dynamics library, 2011. http://www.orocos.org/kdl.

[72] Sanjeev Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stephen Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 639–646. IEEE, 2014.

[73] Mike Stilman and James J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. *Int. Journal of Humanoid Robotics (IJHR)*, 2(04):479–503, 2005.

[74] Mike Stilman, Jan-Ullrich Schamburek, James J. Kuffner, and Tamim Asfour. Manipulation planning among movable obstacles. In *Int. Conf. on Robotics and Automation (ICRA)*, pages 3327–3332. IEEE, 2007.

[75] Moritz Tenorth and Michael Beetz. Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence*, 2015.

[76] Mauro Vallati, Lukáš Chrpa, Marek Grzes, Thomas L. McCluskey, Mark Roberts, and Scott Sanner. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98, 2015.

[77] Jur Van Den Berg, Mike Stilman, James J. Kuffner, Ming Lin, and Dinesh Manocha. Path planning among movable obstacles: a probabilistically complete approach. In *Algorithmic Foundation of Robotics VIII*, pages 599–614. Springer, 2010.

[78] Vincent Vidal. YAHSP3 and YAHSP3-MT in the 8th international planning competition. In *8th International Planning Competition (IPC-2014)*, pages 64–65, 2014.

[79] Rukmini Vijaykumar, Shastri Venkataraman, Gordon Dakin, and Damian M. Lyons. A task grammar approach to the structure and analysis of robot programs. In *Workshop on Languages for Automation*. IEEE, 1987.

[80] Yue Wang, Neil T. Dantam, Swarat Chaudhuri, and Lydia E. Kavraki. Task and motion policy synthesis as liveness games (to appear). In *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2016.

[81] Willow Garage. URDF XML, 2013. http://wiki.ros.org/urdf/XML.