Article

The International Journal of Robotics Research
2018, Vol. 37(10) 1134–1151
© The Author(s) 2018
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/0278364918761570
journals.sagepub.com/home/ijr
SAGE

An incremental constraint-based framework for task and motion planning

Neil T Dantam[1,2], Zachary K Kingston[1], Swarat Chaudhuri[1] and Lydia E Kavraki[1]

Abstract

We present a new constraint-based framework for task and motion planning (TMP). Our approach is extensible, probabilistically complete, and offers improved performance and generality compared with a similar, state-of-the-art planner. The key idea is to leverage incremental constraint solving to efficiently incorporate geometric information at the task level. Using motion feasibility information to guide task planning improves scalability of the overall planner. Our key abstractions address the requirements of manipulation and object rearrangement. We validate our approach on a physical manipulator and evaluate scalability on scenarios with many objects and long plans, showing order-of-magnitude gains compared with the benchmark planner and improved scalability from additional geometric guidance. 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.

Keywords

AI reasoning methods, manipulation planning, path planning for manipulators, task and motion planning

1. Introduction

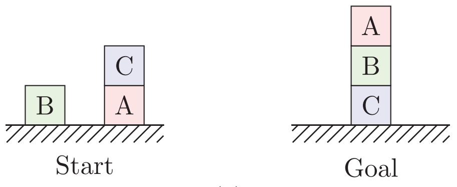
Autonomous robots with complex objectives must reason about both the high-level goals of their task and the geometric motion to execute. The robot must plan over a *task–motion space*, combining discrete decisions about objects and actions with continuous decisions about collision-free paths. Efficient algorithms exist to solve each of these parts in isolation; however, integrating task and motion planning (TMP)[1] 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 *non-existence* of motion plans is difficult and unsolved for the general case (Hauser, 2013b; McCarthy et al., 2012). We directly consider alternate task plans, changing configuration spaces, and motion feasibility to introduce a TMP framework based on incremental constraint solving.

We present a new framework for TMP that offers probabilistic completeness and improved performance and generality over prior work. We first discuss a set of reusable insights on TMP that underlie our approach (see Sections 2 and 4). Our TMP framework extends constraint-based task planning[2] using the incremental solution capabilities of satisfiability modulo theories (SMT) solvers to dynamically incorporate motion feasibility at the task level (see Section 5). We analyze the impacts of modeling on planning scalability and formally prove probabilistic completeness of the algorithm (see Section 6). The incremental framework we employ enables detailed information about motion feasibility to guide the task planner, which we leverage to improve scalability, and we analyze domain assumptions on connectivity that preserve probabilistic completeness (see Section 7). 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 a related TMP method of He et al. (2015) (see Section 8).

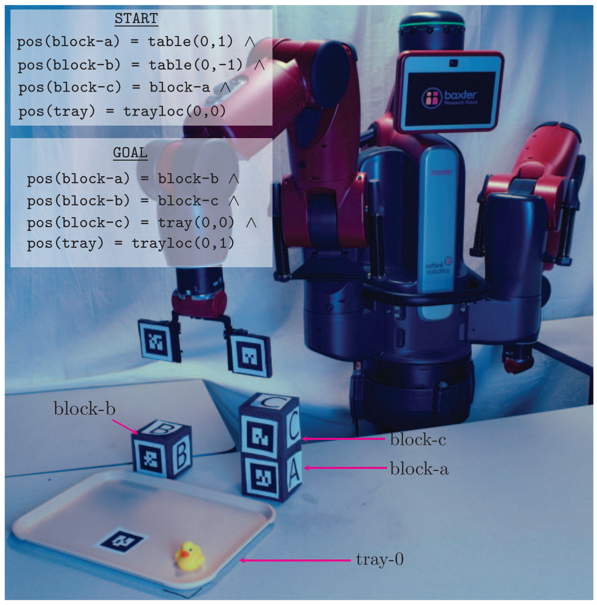
This article extends initial work presented in Dantam et al. (2016b) with a formal analysis of the algorithm, enhanced feedback between the task and motion layers, and additional benchmark results. We analyze how choices in

[1]Department of Computer Science, Rice University, Houston, TX, USA
[2] Current: Department of Computer Science, Colorado School of Mines, Golden, CO, USA

Corresponding authors:
Neil T Dantam and Lydia Kavraki, Department of Computer Science, Rice University, 6100 Main Street, MS 132, Houston, TX 77005, USA.
Email: ndantam@mines.edu, kavraki@rice.edu



(a)



(b)

Fig. 1. A simple TMP problem involving multiple types of actions and goals which must be considered jointly. This problem extends the classic *Sussman anomaly* task planning problem by including the motion of the robot arm and additional actions to push a tray. (a) The Sussman anomaly requires stacking block A on block B and block B on block C. Because the robot cannot move stacked blocks, it cannot treat the goal position of block A and block B independently since that would prevent moving the lower block in the stack. Instead, the full goal condition must be considered together. (b) We extend this classic task planning example to TMP where the robot must also place the blocks on the tray and move the tray, refining each of these discrete task actions into corresponding motion plans. TMP considers the different actions to manipulate the different kinds of objects (transfer blocks, stack blocks, push tray), determines the order of these actions, and identifies collision-free paths for the motion.

modeling the task domain and actions affect state space growth and planning scalability (see Section 6.1), and we formally prove probabilistic completeness (see Section 6.2). We also improve planner scalability by enhancing the feedback between the motion and task layers using collision information from the motion planner (see Sections 7.4 and 8.3). The ability to incorporate such geometric information to guide task planning through additional constraints demonstrates the flexibility of our framework.

2. 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 (Helmert, 2006; Hoffmann and Nebel, 2001; Kautz and Selman, 1999; Rintanen, 2012b) and motion planning (Kavraki et al., 1996; LaValle and Kuffner, 2001) are fundamentally different. Consequently, most TMP methods (Cambon et al., 2009; He et al., 2015; Schüller et al., 2013; Srivastava et al., 2014) 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 to perform efficient and robust TMP.

2.1. Task Planning Requirements

Task planning finds a discrete sequence of actions to transition from a given start state to a desired goal condition (Fikes and Nilsson, 1972).

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.

In addition, 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 (Belta et al., 2005; He et al., 2015; Kress-Gazit et al., 2009), structured and natural language (Kress-Gazit et al., 2008; Matuszek et al., 2013), logical knowledge bases (Erdem et al., 2012; Tenorth and Beetz, 2015), the Planning Domain Definition Language (PDDL) (Gharbi et al., 2015; McDermott et al., 1998; Srivastava et al., 2014), and context-free grammars (Dantam and Stilman, 2013; Lyons and Arbib, 1989; Vijaykumar et al., 1987). 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.

2.2. Motion planning requirements

Geometric motion planning finds a collision-free path from a given start position to a desired goal (Choset et al., 2005). Rearranging objects changes the robot's configuration space, i.e. the valid joint positions. Moreover, some objects are kinematically coupled (e.g. blocks on the tray in Figure 1). Thus, the motion planner must use underlying representations that permit both efficient updates and model object interactions.

2.3. *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 nonexistence of a motion plan (Hauser, 2013b; McCarthy et al., 2012). 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.

3. Related work

3.1. *Task planning*

Task planning is a well-established field, largely evolving from the pioneering work on STRIPS (Fikes and Nilsson, 1972). The currently dominant approaches for efficient task planning (Vallati et al., 2015) are heuristic search (Helmert, 2006; Hoffmann and Nebel, 2001; Vidal, 2014) and constraint-based methods (Kautz and Selman, 1999; Rintanen, 2012b, 2014). Logic programming is also used (Lifschitz, 1999; Tenorth and Beetz, 2015).

Task domains are specified using the PDDL (Edelkamp and Hoffmann, 2004), temporal logics (Belta et al., 2005; He et al., 2015; Kress-Gazit et al., 2009), or formal languages (Dantam and Stilman, 2013). The PDDL, logic, and language forms are closely related and often interchangeable (Cresswell and Coddington, 2004; Dantam and Stilman, 2013; De Giacomo and Vardi, 2000). For the purpose of this work, the choice of notation, e.g. PDDL, temporal logics, grammars, etc., is a matter of convenience rather than a formal necessity. In our benchmark tests, we define the task domain for our implementation with PDDL.

We adopt the constraint-based task planning approach to leverage ongoing advances in solvers for SMT (Barrett et al., 2015; De Moura and Bjørner, 2011). Typical constraint-based planners use Boolean satisfiability formulas (Kautz and Selman, 1999; Rintanen, 2012b). SMT extends Boolean satisfiability with rules (*theories*) for domains such as linear arithmetic. Compared with traditional SAT solvers, SMT solvers provide a more expressive and high-level interface with useful features for expressing constraints in robotics domains (Nedunuri et al., 2014; Wang et al., 2016). Cashmore et al. (2016) compiled PDDL into SMT for task planning in rich domains. Our focus is to leverage the capabilities of SMT solvers to generate alternate plans with guidance from the motion planner. Crucially, some SMT solvers (Barrett et al., 2011; De Moura and Bjørner, 2008) 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.

3.2. *Motion planning*

The major approaches for high-dimensional motion planning are heuristic search (Hart et al., 1968) and sampling-based (Kavraki et al., 1996; LaValle and Kuffner, 2001) methods. Heuristic search planners are sometimes used for manipulation (Cohen et al., 2014), 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 (Kavraki et al., 1996) or single-query trees (LaValle and Kuffner, 2001). 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 (Şucan et al., 2012). Specifically, we use bidirectional rapidly-exploring random trees (RRTs) (Kuffner and LaValle, 2000). 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.

3.3. *TMP*

Most prior work on TMP focuses on performance over completeness or generality. Dornhege et al. (2012) interleaves TMP at the level of individual task actions, calling the motion planner directly from the task planner for feasibility checks using *semantic attachments*. Erdem et al. (2012) produces a knowledge base for household robots in the logic programming paradigm (Lifschitz, 2008). Lagriffoul et al. (2012) applies geometric constraints to limit the motion planning space or prove motion infeasibility in special cases, and Lagriffoul and Andres (2016) uses geometric information to guide the symbolic search. Hierarchical Planning in the Now (HPN) (Kaelbling and Lozano-Pérez, 2011, 2013) interleaves planning and execution, reducing search depth but requiring reversible actions when backtracking. Several related methods (de Silva et al., 2014, 2013a,b; Gharbi et al., 2015) extend hierarchical task networks (HTNs) (Erol et al., 1994) with geometric primitives, using shared literals to control backtracking between the task and motion layer. Srivastava et al. (2014) interfaced off-the-shelf task planners with an optimization-based motion planner (Schulman et al., 2014) using a heuristic to remove potentially interfering objects. Lozano-Pérez and Kaelbling (2014) formulated the *motion* side of TMP as a constraint satisfaction problem over a discretized, preprocessed subset of the configuration space. Nedunuri et al. (2014); Wang et al. (2016) used 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 HTNs (Erol et al., 1994). FFRob (Garrett et al., 2015) developed an FF-like (Hoffmann and Nebel, 2001) task-layer heuristic based on a lazily-expanded roadmap. Bidot et al. (2015) combine forward-chaining task planning with a motion planner to evaluate geometric symbols and backtrack to reconsider previous geometric choices; however, the probabilistic completeness of sampling-based motion planners is not directly addressed. Overall, these methods set aside the general challenge of ensuring probabilistic completeness of the overall approach. In contrast, we directly address the challenge of probabilistically complete TMP.

Other works perform TMP for differential or hybrid systems using sampling-based planners (Karaman and Frazzoli, 2012; Plaku and Hager, 2010; Plaku et al., 2010). Differential dynamics, though necessary for some domains, poses challenges for completeness even in isolated motion planning (Kunz and Stilman, 2015). In contrast, we consider purely geometric motion which is adequate for many manipulation tasks and for which there exist many probabilistically complete motion planners (LaValle, 2006; Şucan et al., 2012).

A smaller number of other task and motion planners also achieve probabilistic completeness. Vega-Brown and Roy (2016) achieve even asymptotic optimality, demonstrating results for planar manipulation. In contrast, our framework focuses on handling high-DOF manipulation domains rather than producing optimal plans. The aSyMov planner (Cambon et al., 2004, 2009; Gravot et al., 2005) combines a heuristic-search task planner based on metric-FF (Hoffmann, 2003) 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 (Cambon et al., 2009). In contrast, we motion plan anew each run, but efficiently update scene data structures to handle object interaction. The Synergistic Framework (Plaku et al., 2010) and related methods (Bhatia et al., 2010, 2011; He et al., 2015) 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. He et al. (2015) also focuses on the manipulation domain. In comparison, our method provides more flexible abstractions than the domain-specific task-state graph used by He et al. (2015), and our method offers significant performance improvements for the benchmarks in Section 8. Thus, we present our new approach as a complementary alternative to prior work on probabilistically complete TMP.

Deshpande et al. (2016) analyzed the decidability of TMP problems, focusing on pick-and-place style domains.

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) (Dogar and Srinivasa, 2011; Levihn et al., 2013; Stilman and Kuffner, 2005; Stilman et al., 2007; Van Den Berg et al., 2010) and minimum constraint removal/displacement (MCR/MCD) (Hauser, 2013a,b) 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 (Krontiris and Bekris, 2015). In contrast to NAMO, MCR, MCD, and rearrangement planning, general TMP considers multiple, arbitrary task actions.

4. 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 Definition 1) and motion domain (see Definition 4), then combine them in the task and motion domain (see Definition 5).

4.1. Task domain

Task domains are typically defined in terms of states and actions using a variety of notations (Erdem et al., 2012; He et al., 2015; Srivastava et al., 2014). 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* $\mathfrak{L} = \left(\mathcal{P}, \mathcal{A}, \mathcal{E}, s^{[0]}, \mathcal{G}\right)$*, where,*

- $\mathcal{P}$ *is the finite state space ranging over variables* $p_0, \ldots, p_n$.
- $\mathcal{A}$ *is the finite 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 finite set of* symbolic transitions, *where* $\mathbb{P}(\mathcal{P})$ *is the power set of* $\mathcal{P}$*. Each* $e_i \in \mathcal{E}$ *denotes transitions* $\text{pre}(a_i) \overset{a_i}{\to} \text{eff}(a_i)$*, where* $\text{pre}(a_i) \subseteq \mathcal{P}$ *is the precondition set,* $a_i \in \mathcal{A}$ *is the operator, and* $\text{eff}(a_i) \subseteq \mathcal{P}$ *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* $s^{[k]} \in \text{pre}(a_i)$*,* $s^{[k+1]} \in \text{eff}(a_i)$*, and for all state variables* p_j *independent of (i.e. free in)* $\text{eff}(a_i)$*,* ${p_j}^{[k+1]} = {p_j}^{[k]}$.
- $s^{[0]} \in \mathcal{P}$ *is the start state.*
- $\mathcal{G} \subseteq \mathcal{P}$ *is the finite 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* $\mathbf{A} = \left(a^{[0]}, a^{[1]}, \ldots, a^{[h]}\right)$*,* $a^{[k]} \in \mathcal{A}$*,* $s^{[k]} \in \text{pre}\left(a^{[k]}\right)$*,* $s^{[k+1]} = a^{[k]}(s^{[k]})$*, and* $s^{[h]} \in \mathcal{G}$.

Note that Definition 1 and Definition 2 are not a new task notation but an abstraction for many existing notations (e.g. Edelkamp and Hoffmann, 2004; McDermott et al., 1998).

4.2. Motion domain

Motion planning algorithms find plans within an abstract *configuration space* ξ (Şucan et al., 2012). The motion plan must follow points within the *free configuration space* ξ_{free}, where $\xi_{\text{free}} \subseteq \xi$. For robot manipulators, the configuration space ξ is typically a real vector space or subset thereof representing joint positions, and the free configuration space ξ_{free} consists of those joint positions not in collision.

Definition 3 (Motion plan)**.** *A* motion plan *is a sequence of neighboring free configurations* $\mathbf{Q} = \left(q^{[0]}, q^{[1]}, \ldots, q^{[n]}\right)$ *such that each* $q^{[k]} \in \xi_{\text{free}}$ *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* $\text{last}(\mathbf{Q}) = q^{[n]}$.

A widely used model for the configuration spaces of robot manipulators is the kinematic tree or scene graph of joints and links (Hartenberg and Denavit, 1964); these structures underlie popular software packages such as OpenRave (Diankov, 2010), KDL (Smits et al., 2011), MoveIt! (Şucan and Chitta, 2015), Gazebo (Koenig and Howard, 2004), and ROS (Foote, 2013).[3] In contrast to prior frameworks, TMP problems involving manipulation 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 Willow Garage (2013) as follows.

Definition 4 (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 local coordinate 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 a configuration 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: ${}^{0}\varsigma_i * \ldots * {}^{j}\varsigma_\ell$, where ${}^{a}\varsigma_b$ is the transform in $\mathcal{SE}(3)$ from parent a to child b and ${}^{\varrho_b}\varsigma_b = \varsigma_b(q)$. Typically, it is not possible to explicitly represent the free configuration space ξ_{free} of high DOF manipulators, so sampling-based planners use specialized collision checkers (Pan et al., 2012) to determine whether a given configuration is valid based on the absolute $\mathcal{SE}(3)$ poses ${}^{0}\varsigma_\ell$ of all geometry μ_ℓ.

The absolute pose of frame f_ℓ in the scene graph is defined recursively as

$$ {}^{0}\varsigma_\ell(q) = \begin{cases} s_\ell(q), & p_\ell = 0 \\ {}^{0}\varsigma_{p_\ell}(q) \otimes s_\ell(q), & p_\ell \neq 0 \end{cases} \tag{1} $$

Common TMP operations such as grasping an object change the free configuration space, which we represent by changing the topology of the scene graph. For example, grasping `block-c` in Figure 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 ${}^{\varrho'_\ell}\varsigma_\ell$ that preserves ${}^{0}\varsigma_\ell$ by ${}^{\varrho'_\ell}\varsigma_\ell = ({}^{0}\varsigma_{\varrho'_\ell})^{-1} * ({}^{0}\varsigma_\ell)$.

While such a tree or graph structure is necessary to model manipulators, mobile robot navigation problems can sometimes use simpler models. Still, we can represent robot navigation instances with a scene graph structure by using only a single frame, rather than a tree, for the robot.

4.3. Task and motion domain

We define the domain $\mathfrak{D}$ for TMP problems in terms of task languages and abstract configuration spaces. Then, we use our scene graphs to model the configuration space. A key detail is the translation between task and motion states and actions, which we define in a *domain semantics* for state abstraction and action refinement. The domain semantics *abstracts* a configuration space to a task state, and it *refines* a task operator to a configuration space and configurations. We define $\mathfrak{D}$ as follows.

Definition 5 (Task–motion domain)**.** $\mathfrak{D} = (\mathfrak{L}, \sigma^{[0]}, \lambda_\alpha, \lambda_\rho, \Omega)$*, where,*

- $\mathfrak{L}$ *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.*
- $\sigma^{[0]} = \left(s^{[0]}, {\xi_{\text{free}}}^{[0]}, q^{[0]}\right)$ *is initial the task–motion state: task state* $s^{[0]}$*, free configuration space* ${\xi_{\text{free}}}^{[0]}$*, and configuration* $q^{[0]}$*.*
- $\lambda_\alpha : \Xi \mapsto \mathcal{P}_m$ *is the state abstraction function that maps free configuration space* $\xi_{\text{free}} \in \Xi$ *to the motion component of task state* $s_m \in \mathcal{P}_m$*.*
- $\lambda_\rho : \Xi \times \mathcal{A} \mapsto \mathbb{P}(\mathcal{Q}) \times \Xi$ *is the action refinement function that maps predecessor free configuration space* ${\xi_{\text{free}}}^{[k]} \in \Xi$ *and task operator* $a^{[k]} \in \mathcal{A}$ *to a motion planning goal (a set of configurations)* $\Theta^{[k]} \subseteq {\xi_{\text{free}}}^{[k]}$ *for the action and a successor free configuration space* ${\xi_{\text{free}}}^{[k+1]} \in \Xi$*.*
- $\Omega \subseteq \Xi \times \mathcal{P}$ *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,* $\left(\Theta^{[k]}, {\xi_{\text{free}}}^{[k+1]}\right) = \lambda_\rho\left(a^{[k]}, {\xi_{\text{free}}}^{[k]}\right)$ *such that* $\text{last}\left(\mathbf{Q}^{[k]}\right) \in \Theta^{[k]} \wedge \text{first}\left(\mathbf{Q}^{[k+1]}\right) = \text{last}\left(\mathbf{Q}^{[k]}\right)$*.*

Now, we use our scene graphs (see Definition 4) to represent the configuration spaces for TMP. We represent the free configuration space at each step, ${\xi_{\text{free}}}^{[k]}$, with a scene graph $\gamma^{[k]}$, and we denote the set of configuration spaces Ξ with a set of scene graphs Γ. The state abstraction function of the domain semantics, λ_α, determines the task–state position $s \in \mathcal{P}$ based on 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 q due to multiple grasp positions or redundancy of the manipulator. The action refinement function of the domain semantics, λ_ρ, determines the subsequent scene graph $\gamma^{[k+1]} \in \Gamma$ for some task operator $a^{[k]}$ via reparenting frames, e.g. to pick and place objects as indicated by the task action $a^{[k]}$.

With TMP formally defined, we proceed to the main contributions of the paper.

5. IDTMP Algorithm

We present a new algorithm for TMP, iteratively deepened task and motion planning (IDTMP). Figure 2 depicts the overall idea of the algorithm to iterate between task planning and motion planning attempts with increasing search depth. Algorithm 1 gives the details of the algorithm. We use a constraint-based task planner to generate candidate task plans (see line 9 of Algorithm 1), then uses a sampling-based motion planner to check plan feasibility (see line 17).

Algorithm 1: IDTMP

```
   Input: $(\mathfrak{L}, \sigma^{[0]}, \lambda_\alpha, \lambda_\rho, \Omega)$ : 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
   /* $\phi$: 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($\phi$);                                   // Push scope
5  $\phi \leftarrow \phi \wedge (\lambda_\alpha(\Omega_\Gamma))^{[h]} \wedge {\Omega_\mathcal{P}}^{[h]}$ ;         // Goal at $h$
6  $\mathbf{T} \leftarrow \emptyset$;                         // T: Task-Motion Plan
7  while $\emptyset = \mathbf{T}$ do
8      $\mathbf{A} \leftarrow \emptyset$;                           // A: Task Plan
       /* Task Planning                                  */
9      while $\emptyset = \mathbf{A}$ do
10         $\mathbf{A} \leftarrow$ Incremental_SMT($\phi$);
11         if $\emptyset = \mathbf{A}$ then /* UNSAT                    */
12             pop($\phi$);                          // Pop scope
13             $\phi \leftarrow \phi \wedge$ (transitions of $\mathfrak{L}$ at step h) ;
14             $(h, t) \leftarrow (h + 1,\ t + \Delta t)$;
15             push($\phi$);                         // Push scope
16             $\phi \leftarrow \phi \wedge (\lambda_\alpha(\Omega_\Gamma))^{[h]} \wedge {\Omega_\mathcal{P}}^{[h]}$ ; // Goal
       /* Motion Refinement                              */
17     foreach $a^{[k]} \in \mathbf{A}$ do
18         $(\Theta^{[k]}, \gamma^{[k+1]}) \leftarrow \lambda_\rho(\gamma^{[k]}, a^{[k]})$;           // Goal
19         $\mathbf{Q}^{[k]} \leftarrow$ motion_plan($\gamma^{[k]}, q^{[k]}, \Theta^{[k]}, t$) ;
20         if $\emptyset = \mathbf{Q}^{[k]}$ then /* Motion Failed   */
21             $\phi \leftarrow \phi \wedge$ (New Constraints) ;
22             $\mathbf{T} \leftarrow \emptyset$;
23             break;  // back to task planner
24         else
25             $q^{[k+1]} \leftarrow$ last($\mathbf{Q}^{[k]}$);
26             $\mathbf{T} \leftarrow$ append($\mathbf{T}, (a^{[k]}, \mathbf{Q}^{[k]})$);
27 return $\mathbf{T}$;
```

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) (Kautz and Selman, 1999) 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.

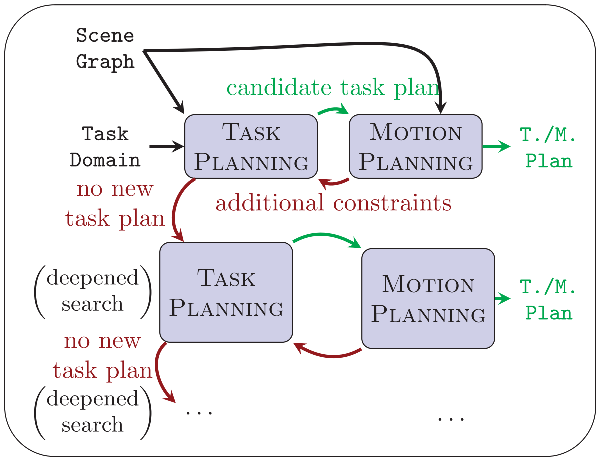


Fig. 2. Diagram of IDTMP. We incrementally incorporate motion feasibility information into the task planner via incremental constraint solving. The bound increases (Kautz and Selman, 1999) of our constraint-based task planner are coupled to timeout increases for sampling-based motion planning.

5.1. *Task planning implementation*

We develop a custom task planner by extending the constraint-based planning method (Kautz and Selman, 1999; LaValle, 2006; Rintanen, 2012a). 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 (Barrett et al., 2011; De Moura and Bjørner, 2008). Our use of incremental SMT solving to update constraints is a new feature compared with previous use of SMT solvers in TMP (Nedunuri et al., 2014; Wang et al., 2016).

5.1.1. 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 (LaValle, 2006, p. 69). 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 $p_0, \dots, p_m$ and actions $a_0, \dots, a_n$, the set of formula variables for h steps is $\left\{ {p_i}^{[k]} \mid i \in 0 \dots m, k \in 0 \dots h \right\} \cup \left\{ {a_j}^{[k]} \mid j \in 0 \dots n, k \in 0 \dots h \right\}$. The formula itself asserts the transitions from Definition 1 and Definition 2. Specifically, the start state holds a step 0 (see line 3 of Algorithm 1) 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, $\left({p_i}^{[k]} = {p_i}^{[k+1]}\right) \vee \left({a_j}^{[k]} \vee \dots \vee {a_\ell}^{[k]}\right)$, where $a_j, \dots, a_\ell$ are the actions that modify p_i. (These are sometimes referred to as *frame axioms*.)
- Only one action is taken at a time: for every action a_i and step k,

$$
{a_i}^{[k]} \implies \left(\neg {a_0}^{[k]} \wedge \dots \wedge \neg {a_{i-1}}^{[k]} \wedge \neg {a_{i+1}}^{[k]} \wedge \dots \wedge \neg {a_n}^{[k]}\right)
$$

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.

5.1.2. 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 with 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. Figure 3 illustrates our use of the constraint stack. 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 (Barrett et al., 2015). In Figure 3, we begin with the start state $s^{[0]}$ on the stack. For each step bound, we push the transition function at that step $f^{[k]}$ onto the stack. The top scope contains the goal at the final step $g^{[k]}$. To increase the step bound (see line 14), we pop goal $g^{[k]}$, and then push an additional transition function $f^{[k+1]}$ and the goal at the increased bound $g^{[k+1]}$. To generate alternate task plans, we push constraint m that incorporates information about motion feasibility (see Section 5.3). To later revisit task plans, we pop m when we increase the step bound.

5.2. *Motion planning implementation*

We use a sampling-based, single-query motion planner, specifically RRT-Connect (Kuffner and LaValle, 2000; Şucan et al., 2012), to instantiate task actions. Using a probabilistically complete, single-query planner backed by our scene graphs (see Definition 4) allows us to ensure probabilistic completeness (see Section 6.2) over changing configuration spaces at the cost of repeated motion planning computation, which we partially address through extensions in Section 7. Alternatively, roadmap-based planners such as

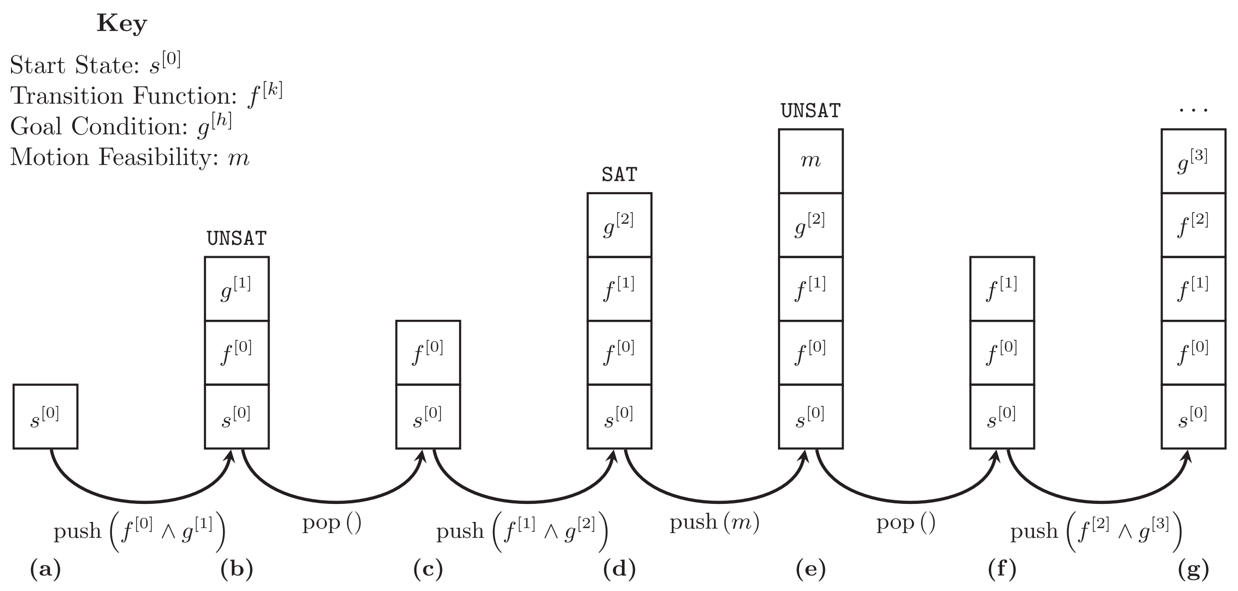

Key
Start State: $s^{[0]}$
Transition Function: $f^{[k]}$
Goal Condition: $g^{[h]}$
Motion Feasibility: m
UNSAT
SAT
UNSAT
...
push $\left(f^{[0]} \wedge g^{[1]}\right)$
pop ()
push $\left(f^{[1]} \wedge g^{[2]}\right)$
push (m)
pop ()
push $\left(f^{[2]} \wedge g^{[3]}\right)$
(a)
(b)
(c)
(d)
(e)
(f)
(g)

Fig. 3. Example illustration of SMT solver constraint stack use by the IDTMP algorithm. (a) Initially, the stack contains only the start state $s^{[0]}$. (b) The transition $f^{[0]}$ from step 0 to step 1 and goal $g^{[1]}$ at step 1 are pushed onto the stack, encoding a plan with one step. (c) No one step plan was found (UNSAT), so the goal at step 1 is popped. (d) The transition $f^{[1]}$ at step 1 and goal $g^{[2]}$ at step 2 are pushed onto the stack, encoding a length 2 plan. A plan is found (SAT), which the motion planner attempts to refine. (e) To generate an alternate task plan, the motion feasibility constraint m (see constraint equations (3), (5), (6)) is pushed onto the stack. (f) No additional two step plan was found (UNSAT), so the motion constraint and goal at step 2 are popped. (g) The transition $f^{[2]}$ at step 2 and goal $g^{[3]}$ at step 3 are pushed onto the stack, and the process continues.

those by Bohlin and Kavraki (2000) and Bohlin and Kavraki (2001) offer the potential for efficient repeated queries. However, in TMP, the scene and configuration space may change with each action taken during the plan, posing challenges for roadmap 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).

5.3. Task–motion interaction

The task–motion interface connects the task layer and motion layer through the *domain semantics* functions λ_α and λ_ρ (see Definition 5). The 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).

We illustrate the use of state abstraction and action refinement functions for the specific example in Figure 1. Initially, the planner applies state abstraction function λ_α to the given starting scene $\gamma^{[0]}$ to produce the starting task state. In this example, the starting task state defines the positions and stacking of the blocks and tray:

$$\begin{aligned} s^{[0]} = \lambda_\alpha(\gamma^{[0]}) = \Big(& (\texttt{position}\,(a) = c) \\ & \wedge (\texttt{position}\,(b) = \ell(2,1)) \\ & \wedge (\texttt{position}\,(c) = \ell(2,0)) \\ & \wedge (\texttt{position}\,(t) = \ell(0,0)) \Big) \end{aligned}$$

Similarly, the planner uses λ_α to compute the task state for the goal scene: $\mathcal{G} = \lambda_\alpha(\gamma_{\text{goal}})$. The start state, goal state, and set of operators define the task language (see Definition 1), providing sufficient information to find candidate task plans. A candidate plan will unstack and restack the blocks and then move the tray:

$$\begin{aligned} \mathbf{A} = \Big(& \texttt{transfer}(c, t(0,0)), \\ & \texttt{stack}(b,c), \\ & \texttt{stack}(a,b), \\ & \texttt{push}(t, \ell(0,1)) \Big) \end{aligned} \tag{2}$$

We use action refinement function λ_ρ to find corresponding motions for each operator in a candidate task plan. For the first operator $\texttt{transfer}(c, t(0,0))$ in the plan above, which moves block c to tray location $(0,0)$, the refinement function λ_ρ will compute the path to move the gripper to a grasping location for c, a path to move the grasped block c to tray location $(0,0)$, and the final scene graph with the transferred block c now placed on the tray. Action refinement continues in this fashion for the rest of the plan. However, if at any point refinement fails, i.e. motion planning times out, then we must find an alternate task plan.

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 give an additional motion feasibility constraint to the task planner to produce the alternate plan (line 21). Figure 3 illustrates how the motion feasibility constraint m is pushed onto the SMT solvers constraint stack and later popped if we must revisit task plans.

The simplest motion feasibility constraint will enumerate plans; more efficient extensions are presented in Sections 7.3 and 7.4. To enumerate plans, the additional constraint asserts that action variable assignments $a_i{}^{[k]}$ be different from that of the previously generated plan:

$$\neg \bigwedge_{k=0}^{h} \left(\bigwedge_{i=0}^{n} \left(a_i{}^{[k]\prime} = a_i{}^{[k]} \right) \right) \tag{3}$$

Then, the next satisfiability check (line 10) will produce a different task plan **A** if one exists.

Since failure of the motion planner to refine operator a_i does not prove a_i is impossible, we later reattempt motion planning for 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 Section 7. 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* (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.

6. Algorithm analysis

6.1. Scalable Task Models

The modeling or encoding of a task domain has an influence on the scalability of planning. Modeling decisions affect the size of the task state space, which, in turn, affects planning performance.

We illustrate the construction of scalable task models with the widely-used `transfer` operator (Alami et al., 1995; Erdmann and Lozano-Pérez, 1987), also referred to as `manipulate` in Stilman et al. (2007). Informally, `transfer` moves a single object from a source object location to a final object location. Though our TMP algorithm maintains independence from any particular operators by separately defining a domain semantics (see Section 4.3), `transfer` is of particular interest due its broad use in domains involving moving or rearrangement of objects.

Constraint-based task planning, which we use and extend, operates on a propositional or *grounded* representation of the state space with variables for the state and selected action (see Section 5.1). The grounded representation "fills-in" the parameters to the first-order logic predicates (representing state) and operators (representing actions) to create Boolean propositions. Thus, the size of the grounded state space, which the constraint solver must search, grows with the number of parameters to predicates and operators.

Two specific representations of `transfer` in PDDL are shown in Figure 4a. Both `transfer` encodings use a finite set of object placement locations and the predicate `occupied`(ℓ), indicating whether an object is already placed at location ℓ. However, differences in action parameters change the growth of the state space for the constraint solver, which in turn changes task planning scalability.

The first `transfer` operator in Figure 4a takes both a source and destination parameter. Location occupancy is modeled with a per-location predicate, `occupied`(ℓ). The effect of `transfer`($o, \ell_{\text{src}}, \ell_{\text{dst}}$) sets `occupied`($\ell_{\text{src}}$) to false and `occupied`($\ell_{\text{dst}}$) to true, where o is the object, ℓ_{src} is the source location, and ℓ_{dst} is the destination location. When we ground this `transfer` operator by filling in both the location parameters, we produce a number of action variables quadratic in the number of locations.

Proposition 1. *The* `transfer` *encoding of Figure 4a produces $O(kn^2)$ grounded action variables, where k is the number of objects and n is the number of locations.*

Proof. One grounded `transfer` action is produced for each of k objects, for each of n source locations, for each of n destination locations, giving $k * n * n$ grounded actions. □

The second `transfer` operator in Figure 4b takes only a single location parameter for the destination. We avoid the need for a source location parameter by using a *derived* predicate `occupied` to determine location occupancy. The first, two-location `transfer` used the operator's effect to set `occupied` for the source and destination locations. Instead, the second, single-location `transfer` determines the location occupancy based on the known locations of each object, i.e. if there exists an object whose location is ℓ, the `occupied`(ℓ) is true. Because we have a separate function for the object's location, `position`(o), we no longer need to include the source location for `transfer`, only the destination to indicate where the object should be placed. When we ground this transfer operator, filling in just the single destination location parameter, we produce a

```
(define (domain transfer-quadratic)
  (:types block location - object)
  (:predicates (at ?obj - block
                   ?loc - location)
               (occupied ?loc - location))
  (:action transfer
           :parameters (?obj - block
                        ?src ?dst - location)
           :precondition (and (at ?obj ?src)
                              (not (occupied ?dst)))
           :effect (and (not (at ?obj ?src))
                        (not (occupied ?src))
                        (at ?obj ?dst)
                        (occupied ?dst))))
```

(a)

```
(define (domain transfer-linear)
  (:types block location - object)
  (:functions (position ?obj - block) - location)
  (:derived (occupied ?loc - location)
            (exists (?obj - block) (= (position ?obj)
                                      ?loc)))
  (:action transfer
           :parameters (?obj - block
                        ?dst - location)
           :precondition (not (occupied ?dst))
           :effect (and (= (position ?obj)
                           ?dst))))
```

(b)

Fig. 4. Two PDDL models of transfer actions. (a) Direct representation of the transfer action that takes as parameters both the source and destination locations of the object. (b) Models transfer with only the destination location as a parameter by using a derived type to determine the `occupied` predicate from object locations. For constraint-based task planners using a grounded encoding, where actions are instantiated for all valid parameter sets, the encoding in (a) will produce a number of grounded actions quadratic in the number of locations while (b) produces only a linear number of grounded actions.

number of action variables that is only linear in the number of locations.

Proposition 2. *The* `transfer` *encoding of Figure 4b produces $O(kn)$ grounded action variables, where k is the number of objects and n is the number of locations.*

Proof. One grounded `transfer` action is produced for each of k objects, for each of n destination locations, giving $k * n$ grounded actions. □

We have used the key `transfer` operator to show how to efficiently model the task domain to reduce the size of the state space that the task planner or constraint solver must search. Eliminating redundant state variables and reducing the number of parameters to operators results in a more compact state space. Because task planning is generally exponential in the size of the state space, reducing state space growth from quadratic to linear strongly impacts planning scalability. Such effects on the state space are important to consider with modeling the task domain.

6.2. Completeness

We now prove the completeness properties of our basic algorithm.

Definition 7 (Probabilistic completeness)**.** *The probability $p(t)$ of finding an existing solution approaches one, increasing monotonically, the more time t is spent computing the solution:*

$$\lim_{t\to\infty} p(t) = 1$$

Theorem 1. *Naïve IDTMP is probabilistically complete.*

Proof. Assume there exists feasible task plan,

$$\mathbf{A} = a_0, a_1, \ldots, a_n$$

where there exists a valid motion plan for each operator a_i. Constraint-based task planning is complete (Kautz and Selman, 1999). At each step horizon n, the task planner enumerates all candidate task plans up to length n by incrementally adding constraint (3). We progressively increment the step horizon n, removing previously added constraints from (3), and will thus identify and revisit task plan $\mathbf{A}$ at all step horizons of n or greater. At a given motion planning timeout t, the probability $P(t)$ of successfully refining $\mathbf{A}$ into its corresponding motion plans is

$$P(t) = p_0(t) * p_1(t) * \ldots * p_n(t)$$

where $p_i(t)$ is the probability of successfully refining operator a_i into its corresponding motion plan. Because the bidirectional RRT motion planner is probabilistically complete (Kuffner and LaValle, 2000),

$$\lim_{t\to\infty} p_i(t) = 1$$

Then, the limit of $P(t)$ distributes over its factors:

$$\lim_{t\to\infty} P(t) = \lim_{t\to\infty} p_0(i) * \lim_{t\to\infty} p_1(i) * \ldots * \lim_{t\to\infty} p_n(i) = 1$$

The probability $P(t)$ of successfully refining $\mathbf{A}$ therefore approaches one as t increases in the limit. □

We have proven probabilistic completeness of naïve IDTMP for scenarios that we can model using the problem formulation in Section 4, and we note the definitional caveat that this analysis does not apply to scenarios that cannot be modeled in this formulation.

7. Completeness-preserving extensions

We now extend the initial IDTMP presented in Section 5 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.

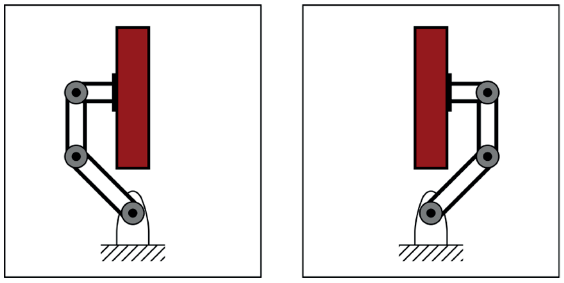


Fig. 5. Example object placements that cause disconnected post-configurations by "blocking-in" the robot. For this 3-DOF planar robot, the two sides of the workspace are disconnected.

7.1. *Connectivity assumption*

In the general case, different plans to move an object may create disconnected configuration-space regions, such as in Figure 5. The basic IDTMP implementation 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 non-existence 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 8 (Connected Configuration Set)**.** *A configuration set* Θ *is* connected *if there is a motion plan between all configurations in* Θ*:*

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

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

Definition 9 (Plan post-configuration)**.** *The plan post-configuration* $\mathcal{E}_{\mathcal{A}\mathcal{Q}}$ *maps from task plan* $\mathbf{A} \in \mathfrak{L}$ *to the set of valid final configurations* $\Theta \subseteq \mathcal{Q}$ *for that task plan,*

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

For example, if we have a plan that ends with grasping some object, the plan post configuration would be the set of all non-colliding inverse-kinematics solutions for valid grasps of the object.

Similarly, the post-configuration of a task operator to place an object is constrained by possible configurations that place the object in the desired destination. This configuration set is disconnected if, as in Figure 5, the placed object blocks the robot from reaching part of the space. We can approximate the plan post-configuration by sampling valid placement configurations. Thus, plan post-configurations are determined by the final operator rather than the entire plan.

Definition 10 (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* $\sigma^{[0]}$*,*

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

For example, consider the block-moving scenario of Figure 1. If we have a final state consisting of several unstacked blocks, there may be many plans leading to this state since we could place the blocks in different orders. Each plan would end with the placement of one of the blocks in its final location. Thus, the state post-configuration would be the union of the placement configurations for each of the blocks.

In general, the state post-configuration $\mathcal{E}_{0\mathcal{Q}}$ is the union of plan post-configurations $\mathcal{E}_{\mathcal{A}\mathcal{Q}}$ for plans satisfying the initial and final condition of $\mathcal{E}_{0\mathcal{Q}}$:

$$\mathcal{E}_{0\mathcal{Q}}(\sigma^{[0]}, s^{[h]}) = \left\{q \in \mathcal{E}_{\mathcal{A}\mathcal{Q}}(\mathbf{A}) \mid \sigma^{[0]} \models \mathrm{pre}(\mathbf{A}) \wedge s^{[h]} \models \mathrm{eff}(\mathbf{A})\right\} \quad (4)$$

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

7.2. *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. $\mathcal{C}(\mathcal{E}_{\mathcal{A}\mathcal{Q}}(\mathbf{A})) = \top$, then the reachable goal configurations are the same starting from all configurations in the plan post-configuration $\mathcal{E}_{\mathcal{A}\mathcal{Q}}(\mathbf{A})$. Consequently, we need consider only a single motion plan for task plan $\mathbf{A}$ to capture all reachable configurations from $\mathcal{E}_{\mathcal{A}\mathcal{Q}}(\mathbf{A})$. Conversely, if the post-configuration is not connected, i.e. $\mathcal{C}(\mathcal{E}_{\mathcal{A}\mathcal{Q}}(\mathbf{A})) = \perp$, 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 $\mathcal{E}_{\mathcal{A}\mathcal{Q}}(\mathbf{A})$.

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 $\mathcal{C}(\mathcal{E}_{\mathcal{A}\mathcal{Q}}(\alpha)) = \top$, 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.

7.3. *Failure-generalization 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 non-existence 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 the non-existence 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.

Constraint (3) requires only that the task planner produce any different plan, pruning only the current, single task plan. Instead, we can prune multiple task plans by generalizing the motion planning failure across multiple steps. We prune plans that attempt a_i from state s for any step from 0 to step horizon h by modifying the incremental constraint added in line 21 of Algorithm 1 with the following constraint asserting that operator a_i not be attempted at state s for all steps:

$$\bigwedge_{k=0}^{h} \left(s^{[k]} \implies \neg {a_i}^{[k]}\right) \tag{5}$$

Then, when we deepen the step and sampling horizons, we likewise pop the added constraint described in (5) so that we will later reattempt these operators with greater search depth. Figure 3 illustrates the pushing and popping of this constraint m.

Using constraint (5), we generalize motion planning failures to all task steps.

7.4. Collision-generalization constraints

While the previous two extensions consider only success or failure of motion planning, we can further extend our framework by considering the specific collisions detected during motion planning. These detected collisions indicate objects that may block the robot, while other objects with which no collision was detected may not affect the feasibility of the current operator. We extend the failure generalization of Section 7.3 to prune *multiple states* based on the locations of these potentially blocking objects.

To identify the potentially blocking objects, we extend the motion planner to track collisions as we compute motion plans to refine each operator. When the collision checker finds that a newly sampled configuration contains a collision, we add the colliding objects to the tracked collision set. If motion planning for the current operator succeeds, the collision set is discarded; otherwise, if motion planning fails, then the collision set indicates the potentially blocking objects for the current operator.

Then, we modify constraint (5), replacing the single s, which constrains the location of *all* objects, with an expression that constrains the locations of only the objects in the collision set:

$$\bigwedge_{k=0}^{h} \left(\left(\bigwedge_{j=0}^{n} {\ell_j}^{[k]} \right) \implies \neg {a_i}^{[k]} \right) \tag{6}$$

where ℓ_j is a proposition indicating that object j is at the location where the collision was detected during the motion planning attempt to refine operator a_i.

The key difference between (5) and (6) is that (5) prunes the failed operator only at the single state where motion planning failed whereas (6) prunes the operator at a *set of states*.

Furthermore, a degenerate case where no collisions are identified during motion planning may occur when a goal pose is outside the robot's workspace or no inverse kinematics solution can be found. In such cases, Equation (6) simplifies to $\bigwedge_{k=0}^{h} \neg {a_i}^{[k]}$, which defers retrying the failed operator a_i for all states and steps at current step horizon h.

Constraint (6) maintains probabilistic completeness under the same assumptions as (5). All states pruned by constraint (6) are those with objects in the locations that impeded the motion planner, and the connectivity assumption ensures that there is no alternate configuration space region to explore. In addition, as with the previous constraint (5), we handle spurious motion planning failures by removing the constraint (6) when we deepen the step and sampling horizons, so as to reattempt motion planning with greater search depth.

The new collision constraint (6) strengthens the connection between the task and motion layers in our framework. Through each of the various motion feasibility constraints, plan enumeration (3), failure generalization (5), and collision generalization (6), we incorporate progressively more precise geometric information into the task planner. The flexibility of the incremental constraint framework enables such additional information to guide the task planner. We see in Figure 3 that only a change to the specific constraint m is required to provide the task planner with further geometric guidance. We show in Section 8.3 that more precise geometric information improves planning scalability.

8. 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 He et al. (2015), 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 with the benchmark planner (He et al., 2015).

Our tests use simulated and physical Rethink Robotics Baxter manipulators. We use Z3 4.3.2 (De Moura and

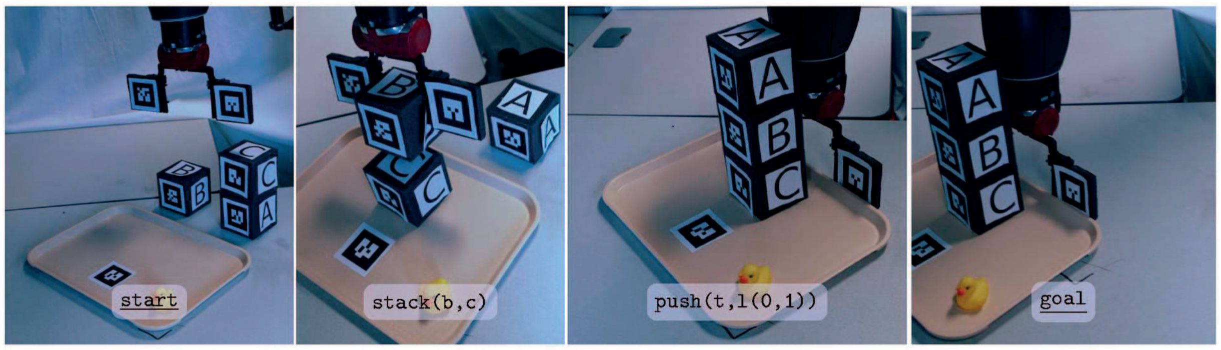
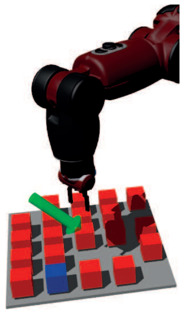
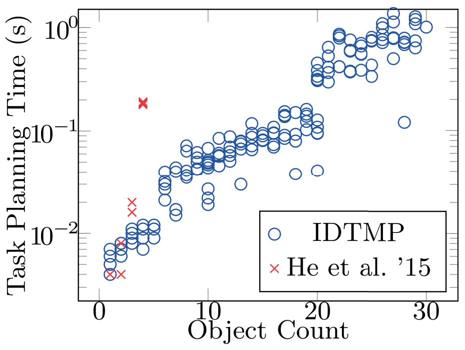
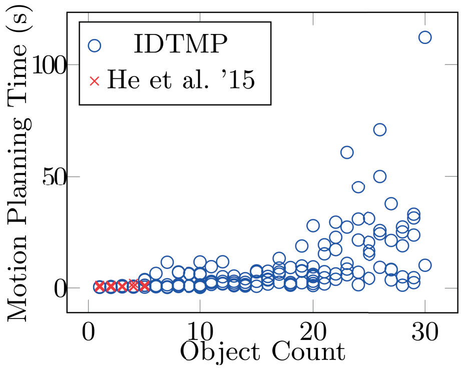


Fig. 6. Physical validation of IDTMP. The robot iteratively plans over actions for transferring blocks, stacking blocks, and pushing the tray and generating collision-free paths for the selected actions. The resulting task–motion plan is successfully executed on the physical robot.


(a)



(b)



(c)

Fig. 7. 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. For each object count, the initial object positions are randomly assigned in a uniform distribution over all locations. (b) Comparison of the task-planning performance of IDTMP and He et al. (2015) over five trials for each object count. (c) Motion-planning times are comparable between both methods. Planning times are cumulative over all iterations, and task times are on a logarithmic scale. Data for He et al. (2015) is only shown up to $n = 5$ due to long task planning times; at $n = 6$, He et al. (2015) took an average of 275 seconds to for task planning.

Bjørner, 2008) as our backend SMT solver, the RRT-Connect (Kuffner and LaValle, 2000) implementation in OMPL (Şucan et al., 2012) for motion planning, FCL (Pan et al., 2012) for collision checking, and POV-Ray (Enzmannn et al., 1994) for visualization. The benchmarks were conducted on an Intel® i7-4790 under Linux 3.16.0-4. We randomly generate the benchmark scenes, resulting in small variance in the results. The physical validation uses the camera registration and servoing method of Dantam et al. (2014) to perform grasping and employs the Ach library (Dantam et al., 2015) and Linux kernel module (Dantam et al., 2016a) for real-time communication. 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.

The task domain for the experiments is based on the single-parameter `transfer` operator discussed in Section 6.1. We create the set of object placement locations using a fixed-resolution grid, though other methods are also possible, e.g. randomly sampled locations or an explicit, user-provided set of locations. The physical experiments also include operators for stacking and pushing.

8.1. *Physical validation*

We validate IDTMP on a physical manipulator for the scenario in Figure 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 Definition 4) 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. Figure 6 shows the robot executing the plan, demonstrating that our overall system works for physical scenarios.

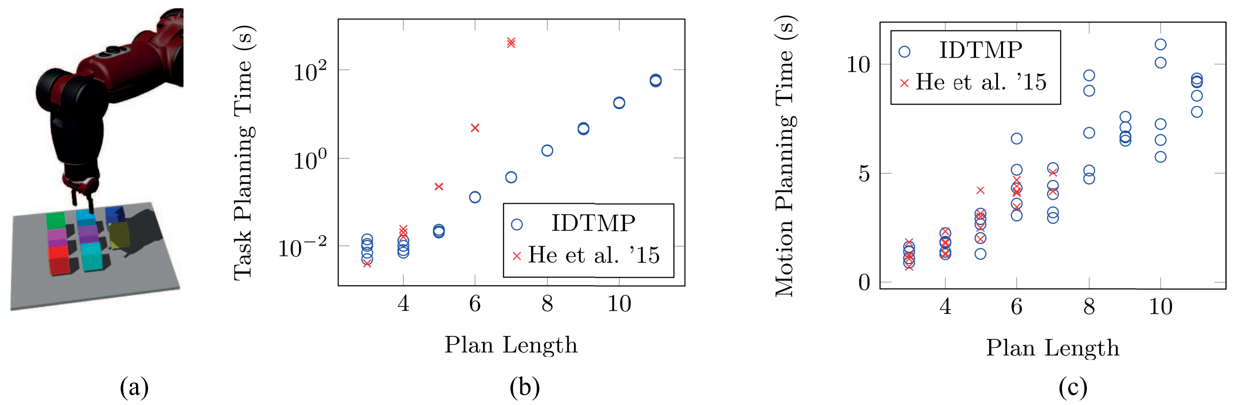


Fig. 8. Scenario testing scalability as necessary plan length increases. (a) The robot must move every block to a different goal location. (b) Comparison of the task-planning performance of IDTMP and He et al. (2015) over five trials for each length. (c) Motion planning times are comparable between both methods. Planning times are cumulative over all iterations, and task times are on a logarithmic scale. Data for He et al. (2015) is only shown up to $n = 7$ due to long task-planning times.

8.2. Scalability tests

We benchmark the scalability of IDTMP against the method of He et al. (2015). For each planner, we measure the time spent in the task planning layer and the time spent in the motion planning layer. For IDTMP, task planning time corresponds to the total time used by the constraint solver, and motion planning time is the cumulative time spent attempting to refine all task operators into motion plans. The total planning time also includes the feedback between the two layers; however, the time for the feedback component was negligible in all cases.

We first test scalability over an increasing number of objects (see Figure 7). While task planning in IDTMP does scale exponentially with the number of objects, it still performs task planning for tens of objects in around 1 second. In comparison, He et al. (2015) scales worse than IDTMP for task planning with increasing object count. Above five objects, task planning using He et al. (2015) took several minutes. Motion planning performance for He et al. (2015) and IDTMP are similar as both use a sampling-based motion planner in the same way to refine candidate task plans. We can see that motion planning times increase with additional objects due to the need to check for collisions between the robot and the added objects. The variance in motion planning times occurs if a motion planning call exceeds the timeout, which is more likely with more objects, and must then be reattempted.

Next, we test scalability for increasing length of the task plan that must be computed (see Figure 8). 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, He et al. (2015) scales worse than IDTMP for task planning in the plan length test. The motion planning time for He et al. (2015) is similar to IDTMP, just as in the object count test. In this case, motion planning scales approximately linearly with increasing plan length. Each additional step in the plan requires an additional motion planning problem of approximately equal difficulty to prior steps, leading to the observed linear scaling.

8.3. Benchmark of IDTMP extensions

Finally, we test the benefit of our completeneness-preserving extensions to IDTMP by rerunning the object count test (see Figure 7a) but with the grasping pose constrained to one side of the object, e.g. when picking from a shelf, resulting in many infeasible operations due to blocking objects (see Figure 9a). Task planning still scales exponentially, though the rate of growth is now improved. However, the informed constraints improve scalability on the motion planning side by reducing the necessary number of motion planning attempts.

Significantly, the collision constraints (6) improve scalability to sub-exponential in the number of blocking objects. TMP is a computationally difficult problem. Quickly pruning infeasible plans is crucial to achieving scalable performance, and the motion planner is able to provide us with strong, though not exact, guidance on plan feasibility. The gains from constraint (6) highlight the incremental constraint framework's flexibility in incorporating additional information from the motion planner.

9. Conclusion

We have discussed the challenges and requirements of TMP, presented a new TMP framework based on incremental constraint satisfaction, and demonstrated scalable planner performance. The TMP requirements and abstractions we have presented in Sections 2 and 4 underlie our planning framework. Moreover, these abstractions model the key features of task–motion domains and may aid the development and analysis of other task–motion planners. Our TMP algorithm is probabilistically complete, handles domains with various actions, and models kinematic coupling between objects.

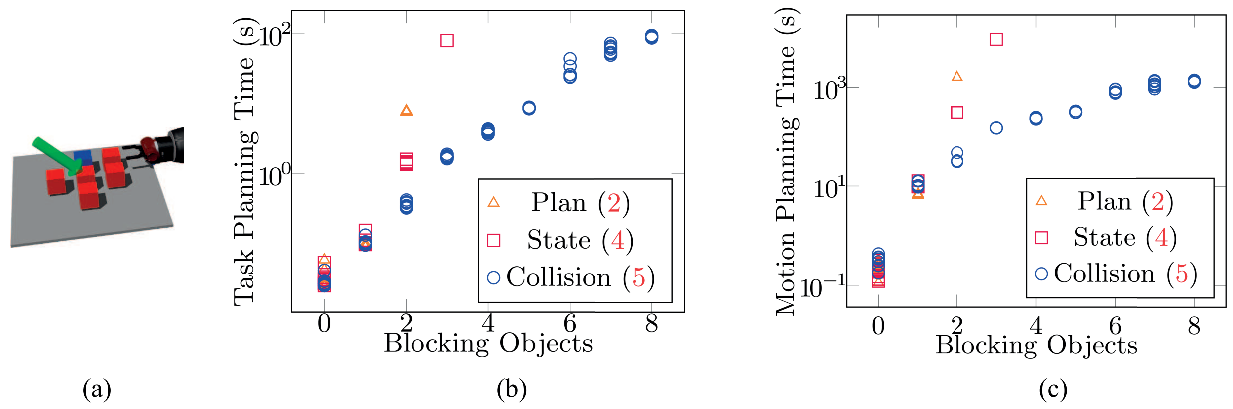


Fig. 9. Comparison of IDTMP extensions. We test planner scalability for the plan enumeration constraint (3), the failure generalization constraint (5), and the collision generalization constraint (6). (a) The object count test (see Figure 7) was rerun with grasp poses constrained to one side of the object, e.g. when picking from a shelf, which causes infeasible actions due to objects blocking the grasp. (b) Task planning time is not significantly changed by the extensions. (c) Motion planning time is improved by the extended constraints. From this semi-log plot, the collision constraint (6) improves scalability of motion planning to sub-exponential in the number of blocking objects. Plan constraint (3) times are only shown up to $n = 2$ and state constraint (5) times to $n = 3$ due to long planning times at larger n.

While the simple form of our algorithm duplicates work at the motion planning level in order to achieve probabilistic completeness over changing configuration spaces, the flexibility of our incremental constraint framework enables increasingly precise information from the motion planner to guide task planning, improving overall scalability. By incorporating collision information into the incremental constraints, we rapidly prune infeasible plans and achieve sub-exponential scalability of the planner.

We have validated IDTMP on a physical Baxter manipulator and have shown that IDTMP provides improved scalability to object count and plan length compared with the manipulation framework of He et al. (2015), a previously developed, similar task and motion planner.

Finally, scaling TMP to handle larger domains remains a challenge. Additional improvements to cache and reuse plans and search trees may help in some cases. Our incremental constraint framework is general to various types of geometric information, and additional constraints based on related methods (Hauser, 2013b; Lagriffoul et al., 2012; Lagriffoul and Andres, 2016; Lozano-Pérez and Kaelbling, 2014; Srivastava et al., 2014) could further aid in pruning infeasible plans and improving overall scalability.

Acknowledgement

We thank Keliang He for his assistance in evaluating He et al. (2015).

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: NTD, SC, and LEK are supported in part by the NSF (grant numbers CCF-1514372, CCF-1162076, and IIS 1317849). ZKK is supported in part by an RUE (associated with IIS 1317849). This work has also been supported by the ERIT program of the K2I Institute at Rice University.

Notes

1. TMP is sometimes also referred to as integrated task and motion planning (ITMP), combined task and motion planning (CTAMP), or task and motion planning (TAMP).
2. *Task* and *constraint* have multiple meanings within planning and robotics. In this article, our usage of "task" corresponds to high-level, logical states or expressions. We do not use the alternate meaning of "task" relating to the Cartesian space or workspace the robot. Our use of "constraint" corresponds to the logical assertions of task planners based on satisfiability checking. We do not use the alternate meaning of "constraint" relating to continuous restrictions limiting robot motion.
3. See also http://wiki.ros.org/urdf/XML

References

Alami R, Laumond JP and Siméon T (1995) Two manipulation planning algorithms. In: *WAFR*. AK Peters, Ltd., pp. 109–125.

Barrett C, Conway CL, Deters M, et al. (2011) CVC4. In: *International Conference on Computer-Aided Verification (CAV)*. New York: Springer, pp. 171–177.

Barrett C, Fontaine P and Tinelli C (2015) The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, http://www.SMT-LIB.org.

Belta C, Isler V and Pappas GJ (2005) Discrete abstractions for robot motion planning and control in polygonal environments. *Transactions on Robotics* 21(5): 864–874.

Bhatia A, Kavraki LE and Vardi MY (2010) Motion planning with hybrid dynamics and temporal goals. In: *Conference on Decision and Control (CDC)*. Atlanta, GA: IEEE, pp. 1108–1115.

Bhatia A, Maly MR, Kavraki LE and Vardi MY (2011) Motion planning with complex goals. *Robotics and Automation Magazine* 18(3): 55–64.

Bidot J, Karlsson L, Lagriffoul F and Saffiotti A (2015) Geometric backtracking for combined task and motion planning in robotic systems. *Artificial Intelligence* 247: 229–265.

Bohlin R and Kavraki LE (2000) Path planning using lazy PRM. In: *International Conference on Robotics and Automation (ICRA)*, volume 1. IEEE, pp. 521–528.

Bohlin R and Kavraki LE (2001) *A Randomized Algorithm for Robot Path Planning Based on Lazy Evaluation*. Dordrecht: Kluwer Academic Publishers, pp. 221–249.

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

Cambon S, Gravot F and Alami R (2004) A robot task planner that merges symbolic and geometric reasoning. In: *European Conference on Artificial Intelligence (ECAI)*, volume 16, p. 895.

Cashmore M, Fox M, Long D and Magazzeni D (2016) A compilation of the full PDDL+ language into SMT. In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling*. AAAI Press, pp. 79–87.

Choset H, Lynch KM, Hutchinson S, et al. (2005) *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA: MIT Press.

Cohen B, Chitta S and Likhachev M (2014) Single and dual-arm motion planning with heuristic search. *The International Journal of Robotics Research* 3(2): 305–320.

Cresswell S and Coddington AM (2004) Compilation of LTL goal formulas into PDDL. In: *European Conference on Artificial Intelligence (ECAI)*.

Şucan IA and Chitta S (2015) MoveIt! Available at: http://moveit.ros.org.

Şucan IA, Moll M and Kavraki LE (2012) The open motion planning library. *Robotics and Automation Magazine* 19(4): 72–82.

Dantam NT, Amor HB, Christensen H and Stilman M (2014) Online camera registration for robot manipulation. In: *International Symposium on Experimental Robotics*. Berlin: Springer, pp. 179–194.

Dantam NT, Bøndergaard K, Johansson MA, Furuholm T and Kavraki LE (2016a) Unix philosophy and the real world: Control software for humanoid robots. *Frontiers in Robotics and Artificial Intelligence* 3: 6. DOI: 10.3389/frobt.2016.00006.

Dantam NT, Kingston Z, Chaudhuri S and Kavraki LE (2016b) Incremental task and motion planning: A constraint-based approach. In: *Robotics: Science and Systems*.

Dantam NT, Lofaro DM, Hereid A, Oh P, Ames A and Stilman M (2015) The ACH IPC library. *Robotics and Automation Magazine* 22(1): 76–85.

Dantam NT and Stilman M (2013) The motion grammar: Analysis of a linguistic method for robot control. *Transactions on Robotics* 29(3): 704–718.

De Giacomo G and Vardi MY (2000) Automata-theoretic approach to planning for temporally extended goals. In: *Recent Advances in AI Planning*. New York: Springer, pp. 226–238.

De Moura L and Bjørner N (2008) Z3: An efficient SMT solver. In: *Tools and Algorithms for the Construction and Analysis of Systems*. New York: Springer, pp. 337–340.

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

de Silva L, Gharbi M, Pandey AK and Alami R (2014) A new approach to combined symbolic-geometric backtracking in the context of human–robot interaction. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3757–3763.

de Silva L, Pandey AK and Alami R (2013a) An interface for interleaved symbolic-geometric planning and backtracking. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 232–239.

de Silva L, Pandey AK, Gharbi M and Alami R (2013b) 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*.

Deshpande A, Kaelbling LP and Lozano-Pérez T (2016) Decidability of semi-holonomic prehensile task and motion planning. In: *Workshop on the Algorithmic Foundations of Robotics*.

Diankov R (2010) *Automated Construction of Robotic Manipulation Programs*. PhD Thesis, Carnegie Mellon University, Robotics Institute. Available at: http://www. programmingvision.com/rosen_diankov_thesis.pdf.

Dogar M and Srinivasa S (2011) A framework for push-grasping in clutter. In: *Robotics: Science and Systems (RSS)*.

Dornhege C, Eyerich P, Keller T, Trüg S, Brenner M and Nebel B (2012) Semantic attachments for domain-independent planning systems. In: *Towards Service Robots for Everyday Environments*. Berlin: Springer, pp. 99–115.

Edelkamp S and Hoffmann J (2004) PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Enzmannn A, Young C and Kretzschmar L (1994) *Ray tracing worlds with POV-Ray*. Sams.

Erdem E, Aker E and Patoglu V (2012) Answer set programming for collaborative housekeeping robotics: representation, reasoning, and execution. *Intelligent Service Robotics* 5(4): 275–291.

Erdmann M and Lozano-Pérez T (1987) On multiple moving objects. *Algorithmica* 2(1-4): 477–521.

Erol K, Hendler J and Nau DS (1994) HTN planning: Complexity and expressivity. In: *AAAI*, volume 94, pp. 1123–1128.

Fikes RE and Nilsson NJ (1972) STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3): 189–208.

Foote T (2013) tf: The transform library. In: *Technologies for Practical Robot Applications (TePRA)*. IEEE, pp. 1–6.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2015) FFRob: An efficient heuristic for task and motion planning. In: *Algorithmic Foundations of Robotics XI*. New York: Springer, pp. 179–195.

Gharbi M, Lallement R and Alami R (2015) Combining symbolic and geometric planning to synthesize human-aware plans: toward more efficient combined search. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6360–6365.

Gravot F, Cambon S and Alami R (2005) aSyMov: a planner that deals with intricate symbolic and geometric problems. In: *International Symposium on Robotics Research (ISRR)*. New York: Springer, pp. 100–110.

Hart PE, Nilsson NJ and Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. *Transactions on Systems Science and Cybernetics* 4(2): 100–107.

Hartenberg RS and Denavit J (1964) *Kinematic Synthesis of Linkages*. New York: McGraw-Hill.

Hauser K (2013a) Minimum constraint displacement motion planning. In: *Robotics: Science and Systems*.

Hauser K (2013b) The minimum constraint removal problem with three robotics applications. *The International Journal of Robotics Research* 33(1): 5–17.

He K, Lahijanian M, Kavraki LE and Vardi MY (2015) Towards manipulation planning with temporal logic specifications. In: *International Conference on Robotics and Automation (ICRA)*. Seattle, WA: IEEE, pp. 346–352.

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

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

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

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

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

Karaman S and Frazzoli E (2012) Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications. In: *American Control Conference (ACC)*. IEEE, pp. 735–742.

Kautz H and Selman B (1999) Unifying SAT-based and graph-based planning. In: *International Joint Conference on Artifical Intelligence (IJCAI)*, volume 99, pp. 318–325.

Kavraki LE, Švestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Transactions on Robotics and Automation* 12(4): 566–580.

Koenig N and Howard A (2004) Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *International Conference on Intelligent Robots and Systems (IROS)*, volume 3. IEEE, pp. 2149–2154.

Kress-Gazit H, Fainekos GE and Pappas GJ (2008) Translating structured English to robot controllers. *Advanced Robotics* 22(12): 1343–1359.

Kress-Gazit H, Fainekos GE and Pappas GJ (2009) Temporal-logic-based reactive mission and motion planning. *Transactions on Robotics* 25(6): 1370–1381.

Krontiris A and Bekris KE (2015) Dealing with difficult instances of object rearrangement. In: *Robotics: Science and Systems (RSS)*.

Kuffner JJ and LaValle SM (2000) RRT-connect: An efficient approach to single-query path planning. In: *International Conference on Robotics and Automation (ICRA)*, volume 2. IEEE, pp. 995–1001.

Kunz T and Stilman M (2015) Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete. In: *Algorithmic Foundations of Robotics XI*. New York: Springer, pp. 233–244.

Lagriffoul F and Andres B (2016) Combining task and motion planning: A culprit detection problem. *The International Journal of Robotics Research* 35(8): 890–927.

Lagriffoul F, Dimitrov D, Saffiotti A and Karlsson L (2012) Constraint propagation on interval bounds for dealing with geometric backtracking. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 957–964.

LaValle SM (2006) *Planning Algorithms*. Cambridge: Cambridge University Press.

LaValle SM and Kuffner JJ (2001) Rapidly-exploring random trees: Progress and prospects. In: Donald B, Lynch K and Rus D (eds.) *Algorithmic and Computational Robotics: New Directions*. A.K. Peters/CRC Press, pp. 293–308.

Levihn M, Scholz J and Stilman M (2013) Hierarchical decision theoretic planning for navigation among movable obstacles. In: *Algorithmic Foundations of Robotics X*. New York: Springer, pp. 19–35.

Lifschitz V (1999) Answer set planning. In: *Logic Programming and Nonmonotonic Reasoning*. New York: Springer, pp. 373–374.

Lifschitz V (2008) What is answer set programming? In: *AAAI*, volume 8, pp. 1594–1597.

Lozano-Pérez T and Kaelbling LP (2014) A constraint-based method for solving sequential manipulation planning problems. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 3684–3691.

Lyons DM and Arbib MA (1989) A formal model of computation for sensory-based robotics. *Transactions on Robotics and Automation* 5(3): 280–293.

Matuszek C, Herbst E, Zettlemoyer L and Fox D (2013) Learning to parse natural language commands to a robot control system. In: *Experimental Robotics*. New York: Springer, pp. 403–415.

McCarthy Z, Bretl T and Hutchinson S (2012) Proving path non-existence using sampling and alpha shapes. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2563–2569.

McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M, Weld D and Wilkins D (1998) *PDDL – the planning domain definition language*. AIPS-98 Planning Competition Committee.

Nedunuri S, Prabhu S, Moll M, Chaudhuri S and Kavraki LE (2014) SMT-based synthesis of integrated task and motion plans from plan outlines. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 655–662.

Pan J, Chitta S and Manocha D (2012) FCL: A general purpose library for collision and proximity queries. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3859–3866.

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

Plaku E, Kavraki LE and Vardi MY (2010) Motion planning with dynamics by a synergistic combination of layers of planning. *Transactions on Robotics* 26(3): 469–482.

Rintanen J (2012a) Engineering efficient planners with SAT. In: *European Conference on Artificial Intelligence (ECAI)*, pp. 684–689.

Rintanen J (2012b) Planning as satisfiability: Heuristics. *Artificial Intelligence* 193: 45–86.

Rintanen J (2014) Madagascar: Scalable planning with sat. In: *8th International Planning Competition (IPC-2014)*, pp. 66–70.

Schüller P, Patoglu V and Erdem E (2013) Levels of integration between low-level reasoning and task planning. In: *AAAI Workshop on Intelligent Robotic Systems*.

Schulman J, Duan Y, Ho J, et al. (2014) Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9): 1251–1270.

Smits R, Bruyninckx H and Aertbeliën E (2011) KDL: Kinematics and dynamics library. Available at: http://www.orocos.org/kdl.

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

Stilman M and Kuffner JJ (2005) Navigation among movable obstacles: Real-time reasoning in complex environments. *International Journal of Humanoid Robotics* 2(4): 479–503.

Stilman M, Schamburek JU, Kuffner JJ and Asfour T (2007) Manipulation planning among movable obstacles. In: *International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3327–3332.

Tenorth M and Beetz M (2015) Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence* 247: 151–169.

Vallati M, Chrpa L, Grzes M, McCluskey TL, Roberts M and Sanner S (2015) The 2014 international planning competition: Progress and trends. *AI Magazine* 36(3): 90–98.

Van Den Berg J, Stilman M, Kuffner JJ, Lin M and Manocha D (2010) Path planning among movable obstacles: a probabilistically complete approach. In: *Algorithmic Foundation of Robotics VIII*. New York: Springer, pp. 599–614.

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

Vidal V (2014) YAHSP3 and YAHSP3-MT in the 8th international planning competition. In: *8th International Planning Competition (IPC-2014)*, pp. 64–65.

Vijaykumar R, Venkataraman S, Dakin G and Lyons DM (1987) A task grammar approach to the structure and analysis of robot programs. In: *Workshop on Languages for Automation*. IEEE.

Wang Y, Dantam NT, Chaudhuri S and Kavraki LE (2016) Task and motion policy synthesis as liveness games. In: *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI.