A NOVEL PERSPECTIVE ON EFFICIENT INTEGRATED TASK AND MOTION PLANNING VIA DIFFERENTIABLE DISTANCE-BASED PREDICATE REPRESENTATIONS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

William Benjamin Thomason

December 2021

© 2021 William Benjamin Thomason

ALL RIGHTS RESERVED

A NOVEL PERSPECTIVE ON EFFICIENT INTEGRATED TASK AND MOTION PLANNING VIA DIFFERENTIABLE DISTANCE-BASED PREDICATE REPRESENTATIONS

William Benjamin Thomason, Ph.D.

Cornell University 2021

Most tasks we want robots to perform require interaction with the physical world via an unknown sequence of parameterized actions; as such, they involve both a continuous geometric component and a discrete symbolic component. For example, a robot cleaning a cluttered surface must reason about how to move objects for cleaning, both over discrete decisions about the order in which to manipulate objects and continuous decisions about grasps and placement locations. Similarly, a robot cooking a meal will need to use a different sequence of discrete actions with continuous parameters (e.g., chopping, stirring, etc.) to follow a recipe, and will need to account for geometric constraints while cooking. Integrated Task and Motion Planning (TMP) is a class of holistic approaches to solving robot planning problems with intricate, interacting symbolic and geometric constraints. TMP offers a promising route to increased autonomy for many real-world robotics tasks, but its requirement for expert-crafted symbolic action models and continued need for higher performance holds back its practical applicability.

This thesis describes work toward addressing these challenges, based on a new framing of TMP as multimodal motion planning guided toward subgoals corresponding to states from which a robot agent can take actions to complete a specified task. We represent these subgoals as formulae composing differentiable functions computing the distance to the nearest state at which a desired property holds. This framing enables us to build two planners which use different approaches to action selection and subgoal

sampling to provide competitive performance on TMP problems while requiring less expert specification as input than alternatives. We further contribute work on automatic symbolic-geometric abstraction repair, a means of easing the specification burden for TMP by allowing initially incorrect or incomplete symbolic abstractions and modifying these abstractions to iteratively converge toward a correct model over time. This work includes an alternative formulation of differentiable-distance-function predicates that offers tight representation of a broader class of non-convex state sets, formally defines the symbolic-geometric abstraction repair problem, and provides a simple anytime bilevel optimization approach to performing abstraction repair from a handful of "unexpected" action execution observations. Finally, this thesis concludes with a brief overview of the current state and incipient directions of TMP research, with an eye toward the problems that remain outstanding for TMP to become practically useful outside of the lab.

BIOGRAPHICAL SKETCH

Wil Thomason was born in Richmond, Virginia. He attended the Maggie L. Walker Governor's School for Government and International Studies in Richmond until 2012, going on to receive his Bachelor's of Science with a dual major in Computer Science and Mathematics from the University of Virginia in May 2015. Wil started his PhD in Computer Science at Cornell University in August 2015, and received his Master's under the advisement of Ross A. Knepper in May 2019. Wil joined Hadas Kress-Gazit's research group in February 2020. Following the completion of his PhD, he will be joining Lydia Kavraki's group at Rice University as a postdoctoral fellow in January 2022.

To my family, for their lifelong support. To Monica, who is always there for the adventures and the late nights, who helped me through deadline crunches and listened to me rant about segfaults, and who is, in all senses, my partner.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1646417 and by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. I am grateful for this support.

I would also like to thank my advisor, Hadas Kress-Gazit, whose support and guidance has been incredibly beneficial to my PhD and my skills as a researcher. I thank my committee for their helpful feedback and review.

I am grateful to my labmates, both in the Verifiable Robotics Research Group and the former Robotic Personal Assistants Lab, for their camaraderie, feedback and research discussion.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Integrated Task and Motion Planning, or "TMP", is a family of approaches to automatically solving complex robot planning problems with richly interacting constraints on both *what* the robot agent does to solve the problem (i.e., its choices of actions) and *how* the agent carries out these actions (i.e., its movements and choices of action parameters). These planning problems, including real-world mobile manipulation, assembly, and surveillance tasks, are core to practical robot utility. TMP has the potential to offer an efficient and scalable path to robot autonomy in these domains, but its requirement for expert-crafted action models and problem specifications holds it back from practical applicability.

This dissertation contributes a novel perspective on TMP combining (1) holistic approaches to planning in a unified state space containing both the discrete and continuous components of the problem state with (2) representations for relational symbols as differentiable functions computing the distance to relation-satisfying states, as well as (3) a conceptual framework and algorithm for automatically constructing and refining symbolic models suitable for planning in this paradigm. Together, these contributions aim to provide a family of flexible, competetively efficient TMP approaches that require less expert effort to use than prior work.

Before presenting these contributions in more detail (Section 1.3 and Chapters 2 to 4), we overview relevant background material on symbolic (Section 1.1.1) and geometric (Section 1.1.2) planning, as well as prior work in TMP (Section 1.2).

1.1 Background: symbolic and geometric planning

Symbolic (or "task") planning and geometric (or "motion") planning are among the oldest, most studied problems in artificial intelligence and robotics. Sections 1.1.1 and 1.1.2 briefly describe the TMP-relevant aspects of each, but a comprehensive overview of either is out of scope of this document. We refer interested readers to Ghallab, Nau, and Traverso [43] or Karpas and Magazzeni [56] for an introduction to and survey of symbolic planning (with a particular focus on applications in robotics), and LaValle [76] for an introduction to and survey of motion planning[1]. Gammell and Strub [35] provide a survey of modern asymptotically optimal sampling-based approaches to motion planning, which relate to the work in Chapter 4.

1.1.1 Symbolic planning

Symbolic planning, often also referred to as "classical AI" planning, is broadly the problem of "combining basic actions to achieve a high-level goal" [56], where said "basic actions" are commonly specified as some manner of discrete or hybrid transition system. This area comprises a diverse range of topics including graph search algorithms [45], planning with temporally extended actions/goals and in temporal modal logics [30, 69, 70], solving (potentially partially observable) Markov decision processes [56], concretely decomposing hierarchical task networks [42][2], and much more [43]. Most TMP work, including this dissertation, focuses specifically on STRIPS-style [32] planning with

[1] However, this reference does not include material on modern motion planning practice or recent advances in motion planning—in particular, on asymptotically optimal motion planning, modern trajectory optimization, and more.

[2] Although some sources separate hierarchical task network planning from classical AI planning, these two planning approaches are functionally similar for the purposes of this work.

discrete, unit-cost actions in a discrete, fully-observable state space. We, as most TMP work, use PDDL [46, 83] to describe symbolic planning domains and problems. Under this regime, a *symbolic planning domain* comprises a discrete transition system over a set of Boolean relations, called *predicates*, on discrete objects, wherein states $s \in S$ are unique combinations of truth values for the possible Boolean propositions formed by applying the domain's predicates to concrete objects. PDDL represents transitions $t_i : S \to S$ as *actions*: schemata parameterized on objects, which specify *preconditions*—propositional logic formulae[3] over states which, if satisfied, allow a corresponding *effect* to take place and alter the set of true propositions, creating a new state. A *symbolic planning problem* adds a set of concrete objects (which grounds the domain's predicates and "lifted" actions to propositions and ground actions via application to subsets of objects) and an initial and goal state ($s_0 \in S$ and $s_g \in S$, respectively). The task of symbolic planning, then, is to find a sequence of transitions $t_1, \ldots, t_n$ which, starting from s_0, both (1) satisfies the recurrence $s_i = t_i(s_{i-1})$ (implying that the precondition of t_i is satisfied by s_{i-1}) for all $1 \leqslant i \leqslant n$, and (2) has $s_n = s_g$ (achieving the goal).

PDDL/STRIPS-paradigm symbolic planning has a vast body of work with many variants on the problem formalism [43]. At a high level, most approaches to STRIPS symbolic planning can be classified as *heuristic-based* or *SAT-based*. Broadly speaking, heuristic-based approaches typically use greedy best-first forward search with respect to a planner-specific action value heuristic. Although early approaches used a "state-space graph" in which nodes corresponded to states and edges to viable actions, later, more successful approaches search a *planning graph*. The planning graph, first introduced by Blum and Furst [12] and subsequently widely adopted and built upon in the symbolic planning community [43, p. 79], is in its basic form a graph with layers of nodes

[3] Different feature variants of PDDL permit different subsets of propositional logic for precondition formulae.

alternating between states (collections of Boolean propositions and associated truth values) and actions. Proposition nodes in the graph have edges to the actions for which they are part of the precondition, and action nodes have edges to the propositions which their effects alter (i.e., make true or false). Further, planners maintain lists of *mutex* actions—actions with incompatible effects—to ensure logical consistency. A dazzling array of approaches to solving symbolic planning with planning-graph-based heuristics and algorithms have been proposed; two of the most significant (and relevant to our consideration) are the FastForward planner of Hoffmann and Nebel [51] and its descendent, the FastDownward planner of Helmert [50].

SAT-based planners, in contrast, transform (through a plethora of analyses and encoding variations) symbolic planning problems to Boolean satisfiability problems, which can then solved via an off-the-shelf SAT solver. Although SAT is known to be NP-complete, modern SAT solvers are often efficient in practice, making SAT-based planning feasible. Kautz, McAllester, and Selman [57] and Kautz, Selman, et al. [59] originally introduced SAT-based symbolic planning. Basic SAT-based planners encode a planning problem by creating a Boolean variable for the value of each proposition at each "step" of the plan (steps are states between action executions), as well as a Boolean variable corresponding to choosing/rejecting each action at each step, and add constraints dictating that (1) actions are viable in a given step only if their preconditions are satisfied in the preceding state, (2) taking an action applies its effect to the state of the following step, changing the set of true propositions accordingly, (3) each step uses exactly one action[4], and (4) proposition values remain constant between subsequent steps unless changed by an action[5]. Finding a satisfying assignment of values to these variables will effectively select a sequence of actions which, when applied, move the system from its

[4] Although there are variants of SAT-based planning which allow for parallel actions.

[5] Also known as maintaining the "frame axioms".

initial state to its goal.

Relation to TMP

"Black box" use of symbolic planners is a common theme in TMP approaches. The TMP planner remains agnostic or mostly agnostic to the specifics of the symbolic planner's method [40, 105, 110, etc.]; avoiding any dependency on specific methods of symbolic planning then allows TMP techniques to improve as the underlying symbolic planners they use improve, "for free". The tradeoff is that TMP planners rely on properties that not all symbolic planners support, such as the ability to generate alternate plans [25]. Further, although symbolic planning is usually not the dominant factor in TMP performance, the choice of symbolic planner can have performance implications [110]. Deciding how (or if) to symbolically encode continuous, geometric state is core to the TMP problem, and the correct choice of encoding may vary with the particulars of the symbolic planner used. Finally, results from symbolic planning provide a lower bound for the computational hardness of TMP: symbolic planning under the STRIPS paradigm is PSPACE-complete on its own [56].

1.1.2 Motion planning

Motion planning is broadly the problem of finding a valid, continuous sequence of robot configurations $q \in \mathcal{Q}$ which move from an initial configuration q_0 in an initial configuration set $Q_0 \subseteq \mathcal{Q}$ to a goal configuration q_g in a goal configuration set $Q_g \subseteq \mathcal{Q}$. The configuration space $\mathcal{Q}$ comprises the variables which suffice to completely describe a robot's pose[6] [76]. Conventionally, these variables are the three-dimensional pose of

[6] For our use. Some configuration spaces encapsulate other information, e.g., velocities or accelerations.

the robot's base (for mobile robots) as well as the variables representing the amount of motion of each non-fixed robot joint along or around its axis of motion. The meaning of "valid" is problem-dependent, but for our purposes (and the purposes of most TMP work) sequence validity implies that all states in the sequence are collision-free and within the bounds of the robot's workspace and joint limits. As with symbolic planning, many variants of the motion planning problem have been studied, including various approaches to optimal planning, planning with consideration of dynamics, planning with constraints beyond collision avoidance and joint limits, planning with uncertainty, etc. [76] We (again, as most TMP work) will restrict our use of "motion planning" to mean the version of the problem operating in a fully observable quasi-static environment (meaning that entities in the environment only move when the robot manipulates them), considering only kinematics (i.e., no dynamics) and collision-avoidance/joint limit constraints. Although there are successful motion planners based on space decomposition and other combinatorial methods [76, chp. 6], most modern motion planning approaches are either *sampling-based* or use *trajectory optimization*.

Sampling-based motion planners rely on the ability to sample (randomly or deterministically) valid configurations to build a graph sparsely representing the valid configuration space, in which nodes correspond to valid states and edges signify a valid motion between two states. A local motion planner constructs the graph's edges—this planner often simply returns the straight-line configuration space path between two states, but may also account for system dynamics, make use of precomputed motion primitives, etc. A full motion plan is then a path through this graph from the node corresponding to q_0 to the node corresponding to q_g. Sampling-based planners vary in the specific kind of graph constructed (e.g., a "roadmap", in which nodes may have multiple connecting paths, or a tree) and the manner in which states are sampled and added to the graph [76, chp. 5]. Most modern algorithms descend either from the Probabilistic Roadmap (PRM)

of Kavraki et al. [60] or the Rapidly-Exploring Random Tree (RRT) of LaValle [77]. Sampling-based planners tend to be *probabilistically complete* which (informally) means that, for any given motion planning problem, the planner will find a solution if one exists with probability approaching one as the number of unique valid samples approaches infinity.

Motion planning with trajectory optimization frames the motion planning problem as an optimal control problem. At a high level, trajectory optimization techniques find valid motion plans by constructing and solving a nonlinear program over a discretized sequence of configurations. This nonlinear program includes constraints formed from the system's dynamics and other validity properties (e.g., collision avoidance) and optimizes a problem-specific base cost function (often trajectory length) [100]. Numerical optimization techniques can then solve the nonlinear program for the desired sequence of valid states and/or control inputs. There are many approaches to trajectory optimization, varying in their formulation of the nonlinear program, method of optimization, and corresponding guarantees and requirements on the system dynamics and cost function to optimize. Some of the techniques most commonly used in robotics include CHOMP [122], TrajOpt [100], and iLQR [80]. Trajectory-optimization-based techniques have the advantage that they naturally consider robot dynamics and produce (locally) optimal solutions, but—compared to sampling-based methods—impose more restrictions on the meaning of state validity and may exhibit lower performance for high-dimensional configuration spaces.

Separate from the method of solving motion planning, TMP generally falls under the umbrella of *multimodal motion planning* [36, 65][7]. In multimodal motion planning, the

[7]This is primarily true for mobile manipulation applications of TMP; others (e.g., surveillance tasks) do not necessarily have a natural multimodal structure.

valid configuration space[8] is subdivided into *modes* corresponding to different submanifolds of valid states. This structure arises naturally in mobile manipulation applications of TMP because we must consider manipulation interactions between the robot and movable objects in the environment; manipulating an object temporarily alters the kinematic graph of the robot, and thus the collision-free valid configuration space. Multimodal motion planning problems are complicated by the restriction to only change modes by taking specific actions at states where the corresponding submanifolds intersect. Finding these states poses a core problem for TMP, as the mode intersections are often of measure zero in the ambient configuration space or in either mode's valid submanifold [36]. Further, as Garrett et al. [36] and others note, many of the problems we would like TMP to solve impose constraints on robot motion (for example, opening a door or drawer), which vary by mode [2, 14, 48, 49]. This dissertation does not consider such constraints, although the methods presented in Chapters 2 and 4 is adaptable to these contexts through the use of a constrained motion planner [63, 64], and the contributions of Chapter 3 are orthogonal to considerations of motion constraints.

Relation to TMP

As discussed, applications of TMP to mobile manipulation—the class of problems primarily considered in this dissertation—can be considered as a form of multimodal motion planning wherein mode transitions occur when applying changes to the valid configuration space corresponding to the effects of symbolic actions. As noted by multiple reviews [36, 72], TMP is differentiated from "pure" multimodal motion planning in that the latter typically cannot account for purely symbolic effects, e.g., the lights in a room turning on or off, whereas TMP can. Motion planning is the most computationally intensive component of most TMP approaches and problems, and finding a means to

[8] The set $Q_v \subseteq \mathcal{Q}$ such that $\forall q_v \in Q_v$, q_v is valid.

reduce the amount of necessary motion planning effort is critical to TMP performance. Neither sampling-based nor trajectory optimization-based planners can prove solution nonexistence in general. This shortcoming means that TMP must consider when to "give up" on a potential motion plan, as well as how to modify the corresponding symbolic solution to attempt to find an alternative. Most TMP approaches are at least partially agnostic to the particular motion planner they use [25, 37, 105], although some [110] are specific to sampling-based motion planning methods and others, most notably Toussaint [112], explicitly frame TMP as a trajectory optimization problem. As with symbolic planning, motion planning is PSPACE-complete on its own [76, chp. 6], which has implications for the computational hardness of TMP.

1.2 An overview of integrated task and motion planning

Garrett et al. [36] provides a comprehensive and (as of this writing) current survey of TMP; this overview will not attempt to duplicate their survey, but will focus on establishing the essential challenges of TMP, highlight some seminal works in the area, and conclude by discussing the advantages of TMP over more naive approaches to solving planning problems with complex, interacting symbolic and geometric constraints. Mansouri, Pecora, and Schüller [82] contribute another recent survey of TMP with a slightly broader scope and higher-level analysis.

1.2.1 The TMP problem

The overall goal of TMP is to automatically solve robot planning problems given as high-level, abstract specifications and requiring the robot agent to choose and execute a

sequence of actions to change the state of the robot and other objects in the environment and achieve the specified goal. TMP specifically targets these problems by combining symbolic planning techniques to choose *what* to do (i.e., which actions) with motion planning techniques to guide *how* to carry out the chosen actions in a physically feasible manner. Although some problems in this broad family involve additional complications such as partial observability and temporally extended actions, we restrict our discussion to the class of problem predominantly considered in TMP work[9] and in Chapters 2 to 4: fully-observable, deterministic symbolic problems in quasi-static environments (as defined in Section 1.1.2).

Under these assumptions, there are five key subproblems for a TMP planner to solve: (1) choosing actions, (2) choosing action parameters, (3) translating between symbolic states and corresponding geometric states, (4) motion planning between states which satisfy the preconditions of the chosen actions for the chosen parameter values, and (5) returning information about motion planner failures to the symbolic problem level to inform search for alternative plans.

Subproblem (1) is often handled by an off-the-shelf task planner [16, 37, 39, 40, 105, 110], although some TMP approaches introduce novel task planning algorithms, usually to incorporate new kinds of geometric constraints in the symbolic planning process [25, 72, 112]. The particulars of the symbolic encoding chosen are crucial for subproblems (1) and (5). The primary tradeoff in encoding is deciding which elements of the problem state should be discretized and which should be left continuous [105]. Many elements of TMP problem state can be represented as either discrete, symbolic variables or variables with a continuous, geometric interpretation. For example, we could encode the state of a manipulator holding an object continuously, by computing force closure [81], or discretely, as a Boolean state variable. Discretizing more of the state,

[9] Although, as discussed in Chapter 5, this focus is starting to change in the field.

e.g., by partitioning a continuous space for object placements into predetermined cells, places more of the burden of planning on the symbolic planner [25] and necessitates additional up-front engineering work to compute an appropriate discretization, whereas forming continuous interpretations for more of the state variables (as we do in Chapters 2 and 4) admits an easier symbolic planning problem at the cost of additional effort in other subproblems [110]. Moreover, while more discretization may lead to a simpler overall problem, it abstracts the problem one step further from physical reality, which may mean that executing a solution requires added work to ensure that the steps are truly physically realizable. Different TMP domains may have different optimal values for this tradeoff.

Subproblem (2) is often one of the most challenging parts of TMP. TMP actions commonly have hybrid parameters: discrete parameters selecting specific objects, manipulators, etc., and continuous parameters selecting properties such as grasps, placement locations, and more. Choices for parameters at earlier steps may affect the correct choices of both discrete actions and continuous parameters at later steps, making the parameter choice subproblem constrained by future actions in the symbolic plan. For example, when solving a pick-and-place problem, the choice of grasp for an object may determine how it or other objects can be manipulated (e.g., by affecting the valid configuration space of the motion planning problem), and thus influence the possible poses in which it may be set down and the feasibility of future pick actions; in a cooking task, the continuous pose of an ingredient on a cutting board may influence how it can be chopped in a later step. Approaches to subproblem (2) vary greatly: in most work, the planner assumes the presence of specialized conditional samplers for action parameters [27, 37, 39, 40, 54], while in other work subproblem (2) is combined with subproblem (3) [65, 110] or another subproblem [112].

Subproblem (3) serves as the "bridge" between the symbolic and geometric prob-

lems. Some planners use a bespoke pair of abstraction and refinement functions as this bridge [25], others symbolic "references" to continuous poses [39, 84, 105], and others still avoid this subproblem by unifying the symbolic and geometric state [110]. Even in this latter setting, some consideration of finding geometric states with given symbolic properties is important for solving subproblem (2).

Subproblem (4) is commonly delegated to an off-the-shelf motion planner [25, 37, 40, 105, 110]. The important aspects of this step are primarily how (or if) the motion planner can cache or otherwise reuse planning effort across invocations, as it will likely be called repeatedly for similar motion planning problems while evaluating alternative symbolic plan and action parameter candidates.

Subproblem (5) is where TMP becomes "Task *and* Motion Planning" rather than "Task *then* Motion Planning". Almost every non-toy TMP problem will require evaluating multiple candidate symbolic plans due to geometric infeasibility. Because the search space of potential candidates is intractably large [43] and each candidate is independently expensive to evaluate, TMP approaches need to use geometric information to order or filter the set of candidates. For many planners, the returned information is solely the step index of a symbolic plan candidate at which motion planning failed [25, 40, 110]. However, some planners make use of more sophisticated geometric constraints to further refine the search space of possible symbolic plans [29, 72, 74, 105].

This decomposition certainly does not perfectly fit all approaches to TMP, as there is a high degree of variation in which subproblems are merged or further split, and different planners tend to focus unevenly on the subproblems. Besides, depending on what problem information is assumed to be given or available, the nature of the "TMP" problems solved by different approaches can vary significantly, perhaps obviating some of these subproblems or introducing others not mentioned here.

1.2.2 Highlights of prior work in TMP

The following is a non-comprehensive review of some significant prior work in TMP, particularly as it relates to the contributions of this dissertation. TMP is an active area of research with a large body of literature including many excellent contributions that this review does not mention. We encourage interested readers to reference Garrett et al. [36] for a more thorough review of current TMP literature, with the caveat that said thorough review still omits some early work in TMP.

As discussed in Section 1.1.2, TMP originates in the multimodal motion planning and manipulation planning literature [8, 36, 47, 72]. Alami, Siméon, and Laumond [2], from these literatures, can arguably be considered one of the first TMP papers. It introduces notions of manipulation modes and presents a framework for manipulation planning in scenes with finitely many object placements and grasps by searching a manipulation graph alternating between "transit" (in which the robot is not holding any object) and "transfer" (in which the robot is holding an object) modes. This "mode graph" paradigm is common to manipulation planning approaches.

Cambon, Alami, and Gravot [16] is another contender for the title, and is the first work we are aware of that specifically combines an off-the-shelf task planner [51] with a motion planner to solve problems with interacting symbolic and geometric constraints. aSyMov abstracts sets of continuous poses as "symbolic positions" [16] corresponding to submanifolds of the free configuration space with different semantic meaning (e.g. feasible grasp poses for a fixed manipulator and object pairing), and builds PRMs [60] incrementally for each mode. Notably, this approach explicitly considers multirobot TMP, which has become less common in the literature since.

Other relatively early work, such as Plaku, Kavraki, and Vardi [91, 92], sits primarily

in the area of pure motion planning, but explores ideas (such as using a discrete search to find a discrete plan that can guide the continuous search for a motion plan) that have become core to modern thinking about TMP.

Plaku and Hager [90] presents an approach to TMP that is closely related in spirit to Thomason and Knepper [110] (Chapter 2). It builds off of earlier work from the linear temporal logic planning community and plans with dynamics using a kinodynamic-RRT-like sampling-based motion planner, which it guides toward task-relevant regions via a symbolic planner for action selection and an assumed "map" function which relates a continuous state to the symbolic predicates that hold at the state.

Kaelbling and Lozano-Pérez [54] takes a new view on TMP by solving problems via best-first action execution. It assumes the presence of specialized black box "geometric suggesters" [54] which compute precondition-satisfying action parameters, and commits to the first action which moves the robot toward its goal by immediately executing it in the real world, before selecting the next action. This aggressively eager evaluation of actions avoids wasted planning effort by deferring expensive parameter searches until they are required by an action which will actually be taken (as opposed to an action in some future state of a candidate symbolic plan), and does lead to an improved ability to efficiently solve large problems with many substeps. However, it also relies on all actions being reversible for completeness, and otherwise may cause the system to reach an unrecoverable state.

Dornhege et al. [27] uses similar ideas to Kaelbling and Lozano-Pérez [54]'s geometric suggesters by proposing *semantic attachments*—efficient "black box" routines which can be attached to a symbolic planner and used to sample and reason about continuous geometric values to solve TMP problems. These samplers are restricted to finite, prediscretized domains.

Although earlier works had touted agnosticism with respect to the particular task and/or motion planners underlying their approaches, Srivastava et al. [105] takes this agnosticism—a common theme in TMP—further by offering a "planner-independent interface layer" for TMP. This interface layer relies on similar symbolic references to sets of continuous values as earlier work [16] and adds predicates over these symbols describing geometric relations to symbolic action descriptions to capture their geometric requirements and effects. Given a task plan with this augmented symbolic model, the interface layer searches for possible satisfying instantiations of the pose references the plan uses. This search is based on a notion of a *constraint network* between steps of the plan to partially address subproblem (2). Viewing parameter selection through the lens of constraint networks is also seen in contemporary [72, 73] and later [39, 40] work. Srivastava et al. [105] also rely on specialized samplers for different types of symbolic reference (e.g., pick poses and place poses).

Lagriffoul et al. [72] also takes a constraint-oriented approach to TMP and focuses on pruning the search space of candidate actions by propagating geometric bounds on continuous action parameters through a constraint network built from a candidate plan under consideration.

Toussaint [112, 113] offers a unique take on TMP, framing the problem as a basic enumerative search over candidate symbolic plans and solving a single nonlinear program (per candidate plan) to choose action parameters and find a motion plan. This formulation is able to handle mode constraints and dynamic manipulation primitives (e.g., hitting, pushing, sliding, etc.) to a better extent than most other approaches, and guarantees global optimality of its solution (as the optimization problem is structured over the entire plan). However, it struggles with long action sequences and requires more modeling effort than other planners, as users must specify continuous models of action constraints

and transition dynamics.

Dantam et al. [25] contributes a performant planner based around incremental adding and removing of symbolic constraints via a modified SAT-based symbolic planner [59]. It is most interesting for its speed, probabilistic completeness (a property relatively few other TMP approaches at the time shared), and use of constraints based on reachability to prune the space of candidate plans. It does not, however, explicitly model constraints between action parameters at different steps, and leaves the translation between symbolic and geometric states to a pair of black-box functions depending on prediscretization.

Garrett, Lozano-Pérez, and Kaelbling [39, 40] are among the fastest TMP solvers today, at the cost of even higher specification effort than alternatives. They combine ideas about conditional samplers and constraint networks into the concept of "streams"—potentially infinite sequences of action parameters of some type, conditioned on input values from other streams, and annotated with predicate relations their inputs must satisfy and that their outputs are guaranteed to satisfy. This formalism, combined in Garrett, Lozano-Pérez, and Kaelbling [39] with a family of algorithms for deferring parameter bindings to different points in the planning process, allows for candidate symbolic plans to be checked incrementally for feasibility by determining if their corresponding streams can return values that are compatible with the constraint network for the entire plan candidate. The downside is that these stream samplers must be additionally specified (in terms of their input/output requirements and properties) and implemented (although in many cases streams can be reused across domains).

Finally, Vega-Brown and Roy [116] is worth highlighting for proving that (given only semialgebraic constraints and uniform stratified accessibility [116] of actions), TMP is no harder than its constituent subproblems—that is, TMP is also PSPACE-complete. This result is notable not only as a relatively rare example of theoretical analysis of the TMP

problem, but in that it would be reasonable to expect that TMP is computationally harder than either task or motion planning, as it often is in practice.

1.2.3 Motivations for TMP

Integrated task and motion planning is worthwhile primarily because of its relevance to desirable applications of robotics—tasks that we usually need a human to solve. Mobile manipulation problems—particularly pick-and-place problems—are the most commonly considered for TMP because they readily admit a multimodal framing and clearly benefit from symbolic planning to guide the sequence of mode transitions [65]. This benefit is greater for problems in domains such as assembly, where the set of actions is more complex and the problems require precise motions to pose parts correctly, or household robotics, where tasks like cooking and cleaning involve careful sequencing of actions with geometric interdependencies (e.g., moving items out of the way on a shelf to clean underneath, washing ingredients before use and keeping them physically separate from unwashed ingredients). Moreover, in these domains (particularly household and construction robotics), although the high level tasks may remain relatively fixed, the particulars of completing the tasks (i.e., action and action parameter choices) will vary significantly based on changes in the environment state (e.g., the locations of tools and utensils).

Neither symbolic nor geometric planning can adequately handle these problems independently. "Normal" motion planners cannot reason about long sequences of actions in a large search space, and even multimodal motion planners require guidance to efficiently find a sequence of mode transitions leading to the goal. Conversely, symbolic planners by their nature operate on an abstraction of a problem. This abstraction necessarily

drops details of the problem, which leads to the generation of symbolic plans which are not physically valid solutions. While a symbolic abstraction's granularity can be increased to capture a greater amount of the problem's physical complexity, this still requires a discretization of the underlying continuous space and has severe implications for symbolic planner performance, rapidly becoming intractable. Further, "task-then-motion planning", in which a symbolic planner generates solutions in isolation, which are then checked for validity by a motion planner, tends to work poorly. Without feedback from the motion planner, the symbolic planner is effectively forced to blindly enumerate possible solutions in an immense search space. Besides, even giving candidate symbolic plans to a motion planner requires a means of translating abstract symbolic states to concrete geometric states, itself a core part of TMP. Efficiently solving problems of this nature fundamentally requires considering the interactions between both forms of constraints, symbolic and geometric.

Finally, TMP techniques improve "for free" with improvements to the underlying symbolic and geometric planning techniques [105]. TMP techniques which frame the problem as one of discovering a sequence of actions to execute synergize well with advances in other areas of robotics, such as recent growth in learning for robot skill controllers [31, 86, 94, 96, 97, 103, etc.]. This synergy provides a potential means of giving learning-based approaches increased generalizability and a capacity for long-horizon planning that they currently lack, while making the planning problem easier. All that is required is a symbolic model for the relevant set of learned skills.

1.3 Contributions

With the background on symbolic planning, geometric planning, and TMP in place, we can return to the contributions of this dissertation in the context of prior work in the field.

The perspective on TMP presented in this dissertation is centered on the idea of planning in a unified state space. This means that the approaches to TMP and symbolic-geometric abstraction repair presented in Chapters 2 to 4 operate in a single state space for both the symbolic and geometric aspects of their problems, and that this state space contains a representation of both the discrete and continuous properties of the problem. Framing the planning space as a hybrid state space is not novel in itself, but even approaches which share this framing view planning as a problem separately in the discrete and continuous parts of the space. Although we do need to consider a projection of this state space to the discrete components (or equivalently the continuous components) at some stages of planning, this projection is a lesser degree of separation between the two types of state than in other approaches. Where other TMP approaches need to translate symbolic states to geometric states (and vice versa, per subproblem (3)), our approaches do not, because all states in the unified space have a symbolic part and a geometric part. For our approaches, the separation between discrete and continuous occurs at the predicate level, such that a given state is associated with a set of discrete facts that are true and a set of continuous properties that hold for its continuous state.

Planning in a unified space directly exposes the multimodal structure of the planning problem and enables several useful themes of the planners in Chapters 2 and 4: reuse of motion planning effort and parallel exploration of multiple candidate task plans without backtracking. Additionally, the unified space framing of TMP is part of what enables our direct method of guiding the search of a sampling-based motion planner as a means of

solving TMP problems.

The other major theme of the work presented in this dissertation is the representation of predicates which are "naturally" continuous as differentiable functions on states, computing the distance from a given state to the closest state that satisfies the predicate's property. The most direct benefit of this choice of representation is that it gives us "for free" efficient samplers for states from which we can successfully execute actions, i.e., without requiring additional annotation or engineering as in other approaches [39, 40, 90]. We are also able to avoid prediscretization by using this approach. Although prediscretization of parameter spaces may improve performance, it limits planner completeness (to resolution completeness [76]) and requires additional up-front manual effort. The tradeoff is that we have less information about the constraints required and imposed by these samplers than a manually engineered stream as in Garrett, Lozano-Pérez, and Kaelbling [39]. This representation also has a beneficial effect on the difficulty of the symbolic planning problems we need to solve: by representing some predicates as functions which we only evaluate when sampling continuous action parameters, we effectively remove them from the symbolic problem, resulting in a relaxed, simplified planning problem. Finally, this distance-based representation is useful beyond the parameter sampling task, as it gives us a guiding signal for motion planning. We can both use this signal to drive the motion planner toward "useful" states for the higher-level plan and (as we do in Chapter 4) defer computationally expensive sampling until we know that its result will be used by the motion planner.

The specific contributions of this dissertation are:

Chapter 2: Introduces the ideas of the unified state space and differentiable distance-function-based representation of geometric predicates, as well as contributing a sampling algorithm which can be used to make off-the-shelf sampling-based motion planners

into competitively performant integrated task and motion planners.

Chapter 3: Contributes an alternative set-based formulation for geometric predicates which admits the requisite distance function computation and sampling properties while offering tighter approximation of non-convex sets. Formally defines the automatic symbolic-geometric abstraction repair problem, and contributes an algorithm for its solution based on a bilevel optimizing search over discrete edits to symbolic formulae, using the aforementioned set-based predicate formulation.

Chapter 4: Builds upon the ideas from Chapter 2 as well as several other elements of the TMP and motion planning literature [25, 105, 106] to contribute an approach to TMP that efficiently finds optimal solutions by deferring action parameter search until it is necessary and conditionally including geometric symbols in the symbolic planning problem to aggressively prune the search space of candidate symbolic plans.

This work builds off of the legacy of multimodal motion planning research [48, 65][10], and extends the line of work in the TMP literature on using specialized samplers to find precondition-satisfying action parameters [27, 38, 54]. It is related to approaches which rely on optimization to find feasible mode transitions [112], as well as TMP approaches which build and connect motion planning structures between modes [16]. Chapter 3 taps into the formal verification and controls communities for its set-based predicate representation, and Chapter 4 in particular builds off of ideas from previous planners [25, 105, 106, 110].

[10] Although Thomason and Knepper [110] slightly predates Kingston et al. [65], many of the ideas in both are similar.

CHAPTER 2

A UNIFIED SAMPLING-BASED APPROACH TO INTEGRATED TASK AND MOTION PLANNING

This chapter is adapted from material previously published as:

> Wil Thomason and Ross A Knepper. "A Unified Sampling-Based Approach to Integrated Task and Motion Planning". In: *International Symposium on Robotics Research (ISRR)*. 2019. URL: https://www.cs.cornell.edu/~wil/papers/isrr2019%5C%5Funifiedtamp.pdf

2.1 Introduction

Generalized robot autonomy requires robots to plan solutions for complex and varied tasks. Most robot problems—for example, manipulation problems—require interaction with the physical world; as such, they involve both a *continuous* geometric component and a *discrete* symbolic component abstracting some part of the problem or world state. Traditionally, these two aspects of the problem are studied independently; motion planners solve geometric problems, whereas task planners solve symbolic problems. If the planners are invoked serially, then the task plan may inform the geometric problem specification. A more holistic planning approach, **integrated task and motion planning** (TMP), seeks to unify the planning problem using information from each component to ease the solution of the other.

The qualitative differences between the task and motion planning problems complicate their effective integration. A task planner operates on a high-level abstraction of the world and thus a valid task plan may have no corresponding valid motion plan if the abstraction is too coarse. A task planning abstraction fine enough to capture the geometric details of the world to the same degree as a motion planner may be intractable

for the task planner. Mitigating this problem by re-running the task planner naively following a motion planner failure is also likely to fail, as most task planners produce only a single plan for a given problem instance. Most TMP approaches focus on creating a better interface between disparate task and motion planners.

This interface often uses the motion planner as a source of constraints and/or a validator for task plans [25, 47, 73, 105]. Using a motion planner as a validator may require repeated executions of both planners, which rapidly becomes expensive. Treating a motion planner as a source of constraints (which is compatible with the validator approach) may be more efficient but is often limited by the manual selection of a restricted set of motion constraints to integrate in the task layer.

We present a novel approach to TMP that fully embeds a symbolic abstraction of a planning problem into the problem's continuous representation. This fused representation (Section 2.3.2) allows a sampling-based motion planner to solve both problems simultaneously and efficiently in a single invocation. To make this process efficient, we also contribute methods for: a factorization of the sampling problem to avoid a dimensional explosion (Section 2.3.2), a continuous semantics for logical formulae in configuration space (Section 2.3.3), and the use of standard task-planning heuristics as sampling biases (Section 2.3.4), which we build into a planner-agnostic sampling algorithm (Section 2.3.5) for solving TMP problems. Finally, we contribute a proof of concept implementation (Section 2.3.6) of our technique and evaluate it (Section 2.5) on a realistic benchmark.

2.2 Related Work

TMP is an active field of research with many existing approaches. Most of these approaches represent task-planning problems using STRIPS-style [32] domains and actions

defined in terms of *Boolean predicates*: representations of symbolic state as Boolean functions applied to environment symbols. We also use this specification format.

Similarly to the "semantic attachments" of Dornhege et al. [27] or the factorized samplers of Garrett, Lozano-Pérez, and Kaelbling [40] (as well as related concepts in other prior work [37, 54]), we rely on user-provided executable predicate semantics implementations to test what predicates a given state satisfies. In contrast to these approaches, our predicate semantics are simple tests on a state's value, do not invoke a motion planner, and are often portable between problems. Further, we use our predicate semantics not only to test known states but also to automatically derive a navigation function to find states satisfying a given predicate.

Prior work has applied sampling-based motion planners to symbolic planning and TMP. Cambon, Alami, and Gravot [16] coupled probabilistic roadmaps (PRM [60]) with symbolic search for transit and transfer paths over a manipulation graph to solve TMP problems, Burfoot, Pineau, and Dudek [15] use rapidly exploring random trees (RRT [77]) to solve STRIPS-style planning problems, and Branicky et al. [14] apply RRTs to planning for hybrid systems (requiring a fusion of continuous and discrete spaces).

Work in multi-modal planning [47–49, 99] and manipulation [7, 8] fuses discrete and continuous configuration spaces. The "modes" of this literature correspond closely to our notion of symbolic state. This work generally assumes the existence of (sometimes precomputed) domain-specific samplers for constraint manifold intersections and uses fixed, specific motion planning algorithms. In contrast, our work contributes an automatic derivation for manifold-intersection samplers (Section 2.3.3) and a sampler compatible with *any* sampling-based motion planning algorithm (Section 2.3.5).

The method of Plaku and Hager [90] is perhaps closest to ours. It also uses a symbolic planner to guide the growth of a random motion planning tree toward a goal. Our approach differs in its handling of purely discrete state (i.e. discrete state without a reasonable continuous representation) and uses a different, improved method of guiding the search and sampling in constraint-satisfying regions.

Approaches to finding constraint-satisfying states are diverse. Dantam et al. [25] use the constraint stack of a satisfiability modulo theories (SMT) solver to efficiently search possible task plans with knowledge of some geometric constraints. Garrett, Lozano-Pérez, and Kaelbling [40] directly sample in feasible regions for actions, whereas Lagriffoul et al. [72] propagate geometric constraints to find such regions, and Toussaint et al. [112, 113] optimize a manually specified differentiable representation of symbolic action constraints. Our approach introduces a continuous semantics for logical formulae specifying the valid execution region of an action and optimizes this automatically derived potential function to locate a state within the region. This approach is also closely related to work in constrained motion planning [64] and formal methods for control synthesis [115].

2.3 Approach

Our approach is simple: we perform normal motion planning in a space including both the geometric and symbolic state for a problem (Section 2.3.2). A valid motion plan in this space will by construction obey all geometric, kinematic, and symbolic constraints; as such, it solves both the task and motion parts of a TMP problem. Motion planning in this space is possible with off-the-shelf sampling-based planning algorithms using our novel sampling algorithm (Section 2.3.5). This sampler guides a planning algorithm toward important regions of the fused geometric/symbolic space without breaking the

planner's other properties, such as completeness and exploration bias.

2.3.1 Geometric and Symbolic Predicates

We consider task-planning problems specified by a *domain* consisting of a set of predicates (Boolean-valued functions) and actions conditioned and operating on these predicates [32, 83]. The "semantics" of a predicate—what it represents in the real world—is determined solely by its use in the definition of the symbolic actions and problem instance. In the context of TMP, we can partition these predicates into two types. *Geometric* predicates can be interpreted in terms of the continuous state space, such as whether one object is on top of another. *Symbolic* predicates have a purely symbolic meaning, such as whether a light is on or off. Each grounding (assignment of ground atoms to arguments) of each symbolic predicate corresponds to one "bit" of symbolic state. Similarly, for each grounding of each geometric predicate there is a corresponding subset of continuous state space where the predicate is true. We call this subset the *predicate region* for a given predicate. Formally:

Definition 1: *Predicates and Predicate Regions*

A Boolean-valued **predicate** is a function $P_i : \mathcal{C} \to \mathbb{Z}_2$, where $\mathcal{C}$ is some configuration space (we define a specific $\mathcal{C}$ below). We define the **predicate region** for P_i to be $T_{P_i} = \{q \in \mathcal{C} \colon P_i(q) = \top\} \subseteq \mathcal{C}$ (that is, the subset of $\mathcal{C}$ where P_i is true). In general, T_{P_i} may be of a lower dimension than $\mathcal{C}$. Given a formula F_i involving one or more predicates, we similarly define a **formula region** T_{F_i}. If the formula is a precondition for an action, we typically refer to its formula region as a **precondition region**.

2.3.2 Composite Space

The hybrid systems, multi-modal planning, and manipulation planning literature commonly combines discrete modes and continuous degrees of freedom into a single planning space [2, 44, 48, 49, 76]. Similarly, we construct a *composite space* in which the "modes" correspond to unique settings of the bits of symbolic state of a TMP problem and the continuous state consists of both robot configuration and movable object pose. Formally,

Definition 2: *Composite Space*

The composite space for a TMP problem is

$$\mathcal{C} = \mathcal{C}_{\text{Robot}} \bigtimes_{M} SE(3) \bigtimes_{B} \mathbb{Z}_2,$$

where M is the number of movable objects, B is bits of symbolic state, $\mathcal{C}_{\text{Robot}}$ is the robot configuration space, and $\mathbb{Z}_2$ is the two-element field. Although we use Boolean symbolic state here, our definitions can be trivially extended to arbitrary symbolic state.

This space may be high-dimensional for even moderate numbers of movable objects and bits of discrete state. Three key insights make planning in composite space tractable: (1) we can factor the composite space to reduce the effective dimensionality for planning; (2) we can avoid sampling in the full object pose space; (3) we can sample within regions that satisfy geometric predicates by introducing an "unsatisfaction semantics". This semantics provides a potential function we can optimize to sample configurations that satisfy arbitrary combinations of predicates. We detail these insights below.

Factoring Composite Space: To decrease the effective dimensionality of a composite space, we can factor it based on unique settings of its bits of symbolic state. This factorization is best understood through a helpful metaphor. By definition, a symbolic predicate describes world state that is not captured by the continuous part of composite

space; it is akin to an additional fact added to the world state[1]. In other words, a particular bit of symbolic state is either true or false for the entire continuous configuration space at a given point in composite space. Thus, we can think about different combinations of values for symbolic predicates as encoding distinct "universes"; continuous configuration spaces where different symbolic facts are true. This factorization permits us to consider a single symbolic state at a time as an otherwise ordinary continuous configuration space, and only need to consider reachable symbolic states (i.e. the combinations of symbolic predicate values attainable by taking some sequence of symbolic actions from the initial symbolic state) rather than directly sampling symbolic states.

Sampling Object Configurations: We assume quasi-static dynamics (i.e. objects move only when manipulated) to further reduce the difficulty of the composite configuration sampling problem. This assumption, which holds for many realistic manipulation problems, allows us to avoid sampling in the full object pose space. Instead, we can track known stable poses of objects (e.g. by adding to a set of valid poses when we choose an action that sets a held object down) and sample from this set. If we have further information about valid stable poses for objects and stable surfaces in the scene, we can sample poses more aggressively, including new poses that match the given template. Our technique neither assumes nor relies on the availability of this information.

2.3.3 Unsatisfaction Semantics

To plan effectively in composite space, we need to efficiently sample states in precondition regions for symbolic actions with effects that move us closer to the symbolic goal. Accomplishing this without either an explicit representation of the regions where a

[1]The distinction between facts captured or not captured by the configuration space is largely a choice of granularity in representation.

geometric predicate holds or a predicate-specific sampler is difficult; all previous work that we are aware of relies on one of these two approaches [9, 40]. We cannot rely on simple uniform sampling of the entire composite space because (1) many geometric predicates correspond to lower-dimensional manifolds (and thus have measure zero in the ambient space) and (2) even geometric predicates that are full-dimensional may be sparsely distributed in the composite space. For a pathological example of this, consider a predicate which is true whenever each of its arguments is an integer. The corresponding predicate region is not a manifold (each of the points where it is true has no open neighborhood), and the integers have measure zero in the reals.

We solve this problem by introducing a continuous semantics for geometric predicates which tells us how far a given state is from the nearest state at which the predicate is true, or how "unsatisfied" it is at that state. A geometric predicate (or formula of geometric predicates) interpreted under this semantics provides a potential function allowing us to project uniform random states from the ambient space directly onto the nearest state in the predicate region. This projection requires computing the **minimal ϵ-weakening** of a predicate at a point.

Definition 3: *ϵ-Weakening*

Take P_i to be a particular predicate defined solely as a formula of the comparison operators $\{=, <, \leqslant, >, \geqslant\} \subset \mathbb{R} \times \mathbb{R} \to \mathbb{Z}_2$, arithmetic operators $\{-, +\} \subset \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, and the Boolean operators $\wedge, \vee, \neg$[2]. Then the *ϵ-weakening* of P_i is defined as $\mathcal{W}(P_i, \epsilon) : \mathcal{C} \to \mathbb{R}$; that is, P_i evaluated with each of the aforementioned comparison operators replaced by

[2] Although we restrict ourselves to this sufficient set of operators in this paper, ϵ-weakening can be extended to allow a broader operator set.

its ϵ-equivalent such that (for a, b real-valued arithmetic expressions):

$$a =_\epsilon b := |a - b| \leqslant \epsilon$$

$$a <_\epsilon b := a < b + \epsilon$$

$$a \leqslant_\epsilon b := a \leqslant b + \epsilon$$

$$a >_\epsilon b := a > b - \epsilon$$

$$a \geqslant_\epsilon b := a \geqslant b - \epsilon$$

We define the **minimal ϵ-weakening** of P_i at a point $q \in \mathcal{C}$ to be $\mathcal{W}_{\min}(P_i, \epsilon)$ for

$$\epsilon = \arg\min_{\epsilon'} \left(P_{i'_\epsilon}(q) = \top\right)$$

We analytically solve for the minimal ϵ for a point $q \in \mathcal{C}$ by refining our ϵ-weakened operators and adding real-valued definitions for the logical operators $\wedge$ and $\vee$ ($\neg$ is implemented as a syntactic transformation, e.g. flipping $\leqslant$ for $>$, etc.):

$$a =_\epsilon b \implies \epsilon = |a - b|$$

$$a <_\epsilon b \implies \epsilon = a - b$$

$$a \leqslant_\epsilon b \implies \epsilon = a - b$$

$$a >_\epsilon b \implies \epsilon = b - a$$

$$a \geqslant_\epsilon b \implies \epsilon = b - a$$

$$a \wedge_\epsilon b := \sqrt{\max(a, 0)^2 + \max(b, 0)^2}$$

$$a \vee_\epsilon b := \min(a, b)$$

We clamp ϵ to $\mathbb{R}^+$, so that $\mathcal{W}_{\min}(P_i, \epsilon)(q)$ defines the shortest distance from a point $q \in \mathcal{C}$ to a point $q' \in \mathcal{C}$ such that $P_i(q') = \top$. We can use $\mathcal{W}_{\min}(P_i, \epsilon)$ as a potential function leading to the predicate region for P_i and use any technique for following potential functions to find a state in said predicate region. For our proof-of-concept

implementation, we use forward-mode automatic differentiation with gradient descent to find a sample in the truth region for any given predicate formula.

This semantics has an important subtlety: if a geometric predicate is combined with other geometric predicates via $\wedge_\epsilon$ or $\vee_\epsilon$ in a formula, then its definition must be restricted to using the above set of operators, conditionals, and iteration in order for the unsatisfaction value of the formula to be correct. Geometric predicates used in isolation in a formula may use *any* operators, etc., so long as (a) they return the result of one of $<, \leqslant, >, \geqslant, =$ and (b) they compute a potential function. This stems from the ϵ-weakening of $\wedge$ and $\vee$; in future work we seek to remove this restriction. In practice, we find the set of allowable predicate functions even under this restriction to be sufficient.

Predicate regions have little required structure. Since $\mathcal{W}_{\min}(P_i, \epsilon)$ gives the distance from a state to the nearest state in the predicate region, $\nabla\mathcal{W}_{\min}(P_i, \epsilon)$ may be discontinuous or have local minima at which P_i is not true. To mitigate these issues, we use smooth approximations of $\min$ and $\max$ in our implementation and standard techniques for escaping local minima. When gradient descent finishes, we test the resulting state q against the ordinary Boolean version of P_i to ensure that it is valid. If it is not (i.e. $P_i(q) = \texttt{false}$), we repeat gradient descent from a new uniform random sample in $\mathcal{C}$.

2.3.4 Heuristic Guidance

We need to be able to pick precondition regions as subgoals to efficiently plan in composite space. In other words, we require a means of selecting symbolic actions which will accomplish the goal. We handle this by defining the following minimal interface to a task-planning heuristic:

Definition 4: *Heuristic interface*

A heuristic $\mathbb{H}(q)$ must return a list of actions $\vec{A}$ for which the symbolic part of state q satisfies the symbolic part of the precondition of each $a \in \vec{A}$. $\mathbb{H}(q)$ may optionally sort $\vec{A}$ by $\texttt{Priority}(a)$ for each $a \in \vec{A}$.

Many heuristics and other methods of action suggestion satisfy this interface. In our proof of concept implementation, we use the "helpful actions" heuristic from the FF planner [51], chosen for its simplicity. The heuristic provides long-horizon symbolic planning guidance; the quality of this guidance depends on the particular heuristic chosen. The meaning of `Priority` depends on the particular heuristic; intuitively, it estimates how important the action is to the overall solution.

We make no further assumptions on the heuristic. This means that we cannot assume the heuristic has any knowledge of the geometry of the scene, the kinematics of the robot, or any other factors that inform the physical feasibility of a symbolic action. This information is critical for solving TMP problems efficiently. As such, we define the *scaled priority* of a symbolic action to be:

$$\texttt{P}(a) = \frac{\texttt{Priority}(a)}{1 + \texttt{Fail}(a) + \gamma * \texttt{Success}(a)}$$

where $\texttt{Priority}(a)$ is the heuristic priority of action a, $1 \leqslant \gamma \in \mathbb{R}$ is a constant scaling factor, and $\texttt{Fail}(a)$ and $\texttt{Success}(a)$ are counts of the number of attempts to use a that have failed (i.e. we could not find a state satisfying the precondition, or the state satisfying the precondition was not added to the planner's data structure) and succeeded (i.e. we found a state satisfying the action precondition and added it to the planner's data structure), respectively. Intuitively, if an action keeps failing, it is less likely to be part of a valid plan and should be deprioritized; if an action succeeds, then it should also be deprioritized to avoid using the same actions redundantly.

2.3.5 Composite Space Sampling

We combine the tools from the previous sections into a sampling algorithm, shown in Algorithm 1. Any sampling-based motion planner can use this algorithm as its sampler to solve TMP problems in composite space (with the caveat that bidirectional planners require further adaptation to plan in composite space, where "distance" is directional).

We use the heuristic of Section 2.3.4 to choose useful actions to attempt and use ϵ-weakening to efficiently sample in the precondition regions for these actions. At a high level, our algorithm treats these precondition regions as sub-goals for a sampling-based motion planner. By planning between sub-goals in sequence, we are able to construct a chain of motion plans which corresponds to a complete TMP solution. The call to the heuristic on Line 16 of Algorithm 1 has one important subtlety: we request viable actions from the heuristic for a particular symbolic state only *after* we have reached it; as such, we avoid unnecessary task-planning effort by never forcing the heuristic to do work for an impossible symbolic state.

The calls `UpdateInfo` and `UpdateLog` update planner metadata. `UpdateLog` logs associations between configurations and symbolic actions. This log is used to instruct the robot to run the controller for a specific symbolic action at the correct state when executing a plan. `UpdateInfo` updates a planner data structure tracking the valid object pose sets associated with a symbolic state as well as symbolic state connectivity. This information is used during sampling and distance computation.

[3] Using a biased coin ensures that we sometimes grow the planning structure "normally" within single symbolic configurations. β may be thought of as trading off between the motion planning problem and the symbolic planning problem.

Algorithm 1: Composite Space Sampling

Data: Task specification, scene description, predicate semantics implementations
Sampler Context : For $c \in \mathcal{C}$ and $u \in U$: a log of symbolic actions (if any) taken in c; a set of metadata associated with u including valid object poses, etc.
Output : Sampled state in $\mathcal{C}$

```
// Flip a coin which returns True with probability β[3]
if BiasedCoin() is True then
    return NormalSample()
else
    return HeuristicSample()
function NormalSample()
    u ← UniformRandom(U);                    // Symbolic state
    p ← UniformRandom(ValidPoses(u));  // Known valid pose
    r ← UniformRandom(C_Robot);        // Robot configuration
    return MakeConfiguration(r, p, u);      // Form composite
     config
function HeuristicSample()
    repeat
        c' ← NormalSample();
        // Choose a valid action for the sampled
           symbolic state:  First, get the set of
           actions.
        // This corresponds to ℍ(q) of Definition 4
        h ← PrioritizedActions(c');
        // Sample an action according to its scaled
           priority
        a ← PrioritySample(h);
        f ← Precondition(a);
        // Solve for a precondition-satisfying state
           using a black-box optimizer
        c ← Solve(c', f);
    until Satisfies(c, Precondition(a));
    UpdateInfo(c.u, GetPoses(c)); // Add to c.u viable pose
     set
    UpdateLog(c, a);          // Log use of a at c for plan
     execution
    return c;
```

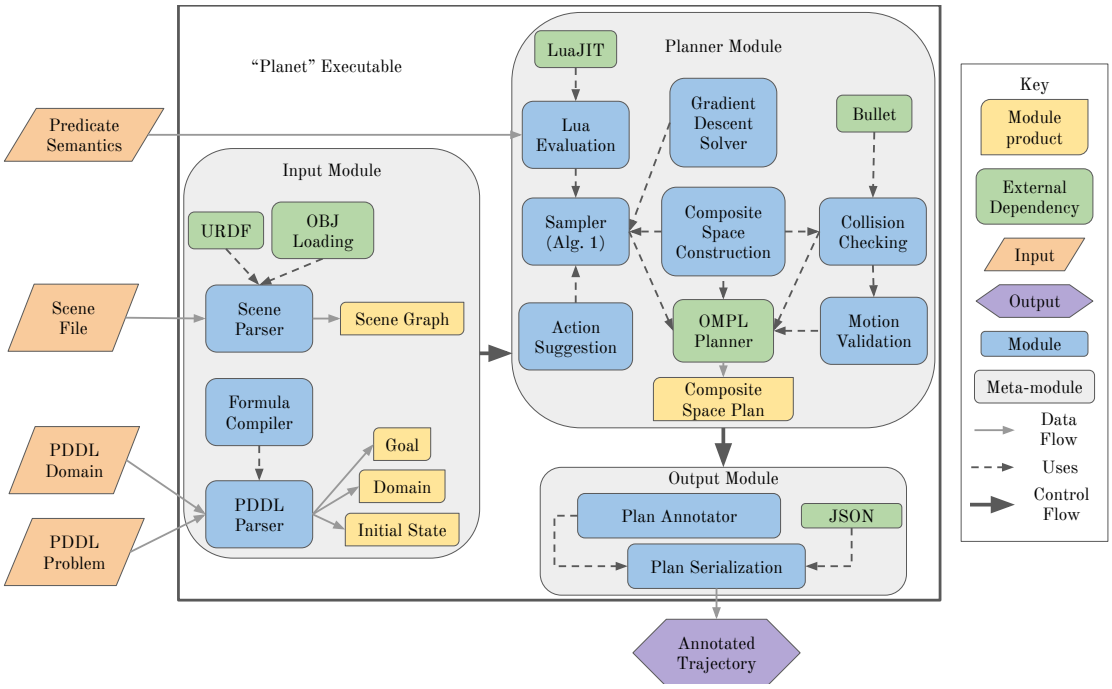


Figure 2.1: The architecture of our proof of concept implementation. We take as input a PDDL domain and problem, as well as a file describing the robot and workspace and Lua semantics for a subset of the PDDL predicates. We use these inputs to construct a composite space for the problem and run a standard OMPL [108] motion planner in this space with our sampling algorithm (Algorithm 1). The resulting composite state plan is transformed into a JSON representation of a robot pose trajectory annotated with optional actions (e.g. grasping) to take at each point in the trajectory. This trajectory can be executed by a standard controller for the robot.

2.3.6 Implementation

We have implemented a proof of concept of our algorithm in C++. Fig. 2.1 shows our system architecture. We use OMPL [108] for motion planning (with our custom sampler and composite configuration space as described in Section 2.3) and Bullet [23] for geometric collision checking. We use OMPL's implementation of RRT [77] as our sampling-based motion planner. We use PDDL extended with keywords to signify kinematic and geometric predicates to specify symbolic domains and Lua functions to specify predicate semantics; Listing 1 shows an example predicate definition in Lua. We chose Lua for its speed (via LuaJIT), easy interoperability with C++, and the availability of high-quality scientific computing libraries for automatic differentiation. In particular,

we use SciLua for automatic differentiation.

```
function above(x, y)
  return And(Le(x.pz - y.pz, 0.15), Ge(x.pz - y.pz, 0.0))
end
```

Listing 1: The Lua implementation of a predicate describing one object being above another.

We specify problems as a PDDL domain and problem instance coupled with a scene file in the format described in Lagriffoul et al. [71]. We use this format for the geometric part of our problem specifications to make it easier to benchmark our planner against future planners also using this format. Source code for our planner is available at https://github.com/wbthomason-robotics/planet.

2.4 Analysis

The bias β of the coin flip used in Algorithm 1 to choose between "normal" and heuristic sampling is tunable and may have different optimal values in different domains. Both normal and heuristic sampling are necessary for efficient performance; heuristic sampling is more expensive and only returns states in action precondition regions, whereas the more efficient and completeness-preserving "normal" sampling may be unable to find some precondition regions and thus cannot grow the tree between symbolic states.

The minimal heuristic interface defined by Definition 4 allows us to swap out one action suggestion method for another without having to change our technique at all. For instance, the proof of concept implementation discussed in Section 2.3.6 implements the FF [51] "useful actions" heuristic, but we could equally easily use a neural network policy trained on a particular planning domain, a more sophisticated planning graph analysis heuristic, or even some combination of multiple heuristics or a full task planner.

2.4.1 Probabilistic Completeness

Algorithm 1 preserves the probabilistic completeness of the motion planning algorithm using it as a sampler (so long as `BiasedCoin` returns both `True` and `False` a non-zero fraction β of the time and the action suggestion method eventually returns all feasible actions for a given state). The proof is straightforward; we sketch it here:

Proof. As stated above, assume that `BiasedCoin` returns `True` with probability $0 < \beta < 1$. Take $\mathbb{H}$ to be the action suggestion method used by Line 16 of Algorithm 1 and further assume that for any state $q \in \mathcal{C}$,

$$\bigcup_{n \to \infty} \mathbb{H}(q) = \texttt{ViableActions}(q)$$

where $\texttt{ViableActions}(q)$ is the set of all actions which have precondition regions in the symbolic state of q, and $\bigcup_{n\to\infty} \mathbb{H}(q)$ is the union of the sets of actions suggested by $\mathbb{H}$ over n invocations of $\mathbb{H}$ for q as n goes to ∞[4]. In other words, $\mathbb{H}$ always eventually suggests all viable actions for a state $q \in \mathcal{C}$.

Let i be the number of sampled states and S the set of samples. As $i \to \infty$, also $n \to \infty$, and thus $\forall q \in S$, $\mathbb{H}$ eventually suggests every viable action from q. Thus, given Algorithm 1, as $i \to \infty$, every viable action a for a sampled $q \in \mathcal{C}$ will be suggested and attempted infinitely often. Each invocation of the black-box solver starts from a uniform random configuration; as the "cost" optimized by the solver (distance to a precondition region) is convex, this means that as $i \to \infty$ dispersion in the precondition region [75] will go to zero (we effectively project from a ball around the precondition region into the precondition region). Performing a viable action a from q introduces

[4]For deterministic, stateless $\mathbb{H}$, the set of actions returned will be the same on every invocation. If $\mathbb{H}$ tracks the set of actions already suggested or is nondeterministic, then the set of actions returned may differ on subsequent invocations of $\mathbb{H}$ for q.

symbolic states and object poses caused by applying the effect of a to q. Thus, as $i \to \infty$, we will eventually find precondition-satisfying samples arbitrarily close to every precondition-satisfying state, and thus we will eventually discover every "relevant" object pose set and symbolic state.

Finally, as $i \to \infty$, `NormalSample` will be invoked infinitely many times. As `NormalSample` amounts to uniform random sampling of robot configurations and known object pose sets and symbolic states, and as we have established, every object pose set and symbolic state which could be part of a solution to the TMP problem will eventually be discovered, it follows that `NormalSample` will eventually sample within a ε-neighborhood of every state which could be a part of a solution to the TMP problem. Hence, if the planning algorithm using the sampler defined by Algorithm 1 is probabilistically complete using a "normal" sampler alone, it will remain probabilistically complete when solving TMP problems using Algorithm 1 as its sampler. □

2.5 Evaluation

We evaluate our proof of concept implementation on two benchmark problems from the set proposed by Lagriffoul et al. [71]. We modify the "clutter sorting" problem by adding our required annotations of kinematic and discrete predicates and providing an implementation of the predicate semantics. We show results on this variant of the problem as well as a variant with the red blocks removed from the scene. We include this second variant to show that our system can handle obstacles like the red blocks when they are present and also improve its performance in their absence.

We have chosen to use the problem format proposed by Lagriffoul et al. [71] for our evaluation in an effort to make it easier to evaluate future TMP systems against ours

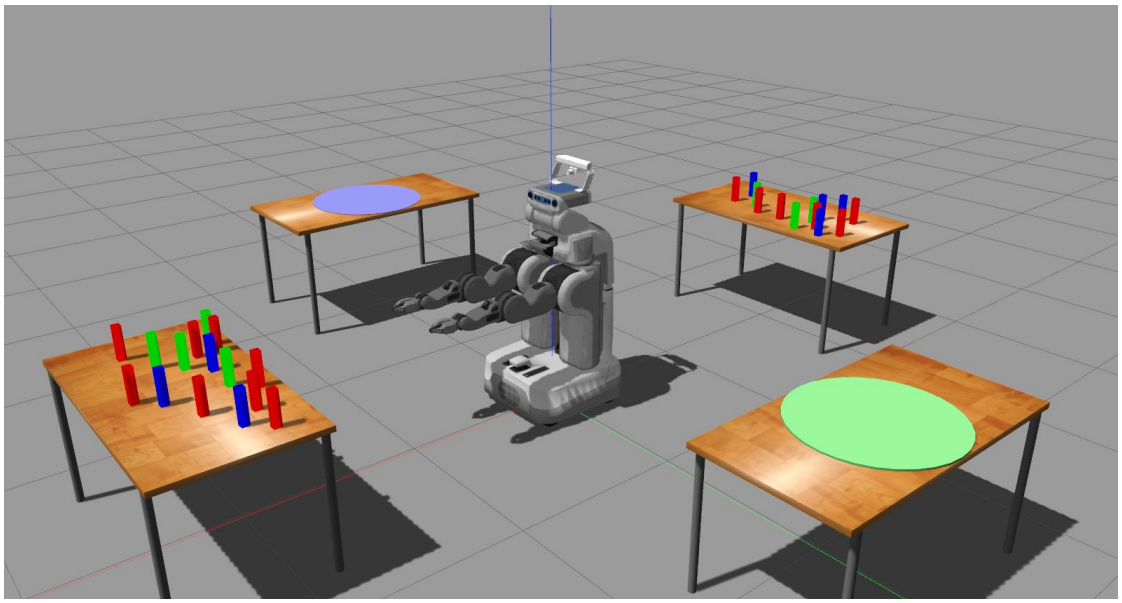


Figure 2.2: Our simulation environment for the benchmark problem discussed in Section 2.5.1. The PR2 must find a plan to move the green blocks to the table with the green circle and the blue blocks to the table with the blue circle.

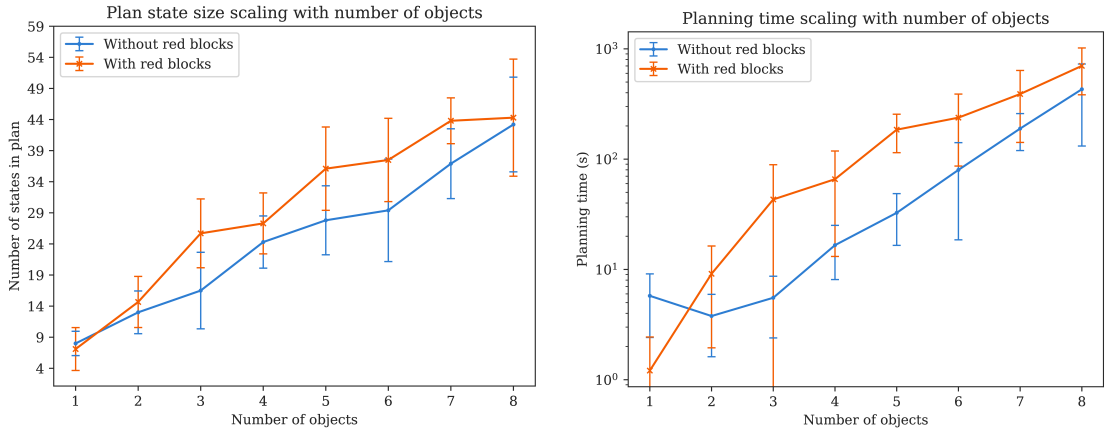
(i.e. by using what is intended as a standard specification format and set of benchmark problems for the field).

2.5.1 Scenario Description

Our variant of the clutter problem of Lagriffoul et al. [71] involves sorting two groups of blocks into specific zones located out of robot reach from each other. This problem demonstrates scalability with the number of objects in the scene as well as the capability to handle infeasible task actions in a large task space. The problem, shown in Fig. 2.2, specifies a Willow Garage PR2 robot and places the blocks on a set of four tables.

2.5.2 Experimental Results

We have run our implementation on clutter problem instances ranging in size (the number of objects to be moved) from one to eight, for ten trials of each instance size. The objects to be moved are split evenly among blue and green blocks; if the total number is odd, there is one more blue block than green block. There are always either zero (the blue line in Figs. 2.3a and 2.3b) or fourteen (the orange line in Figs. 2.3a and 2.3b) red blocks present in the environment. Figure 2.3a shows the number of states in the solution, and Fig. 2.3b shows the planning time in seconds. We plot the mean time for each instance size. The error bars show one standard deviation from each mean.



(a) *(b)*

Figure 2.3: Performance results on the "Clutter" problem

Fairly benchmarking TMP planners is challenging. In addition to the usual problems with reporting wall-clock time (different computer speeds, implementation languages, levels of engineering "polish" and optimization), the results of raw performance comparisons depend heavily on choices made in the problem specification. For example, the specification makes choices about whether variables should be discretized or left in the continuum, how the semantics of the problem are implemented as predicates, and even what state is relevant to the problem. These factors (and more) may affect the

performance of different TMP planners in different ways. Further, wall clock time may not capture precomputation or other planning effort conducted offline; our technique specifically avoids this sort of offline work. This field is still maturing, and it has taken only first steps at standarizing the TMP problem [71]. The community still does not have consensus on a good set of performance metrics to compare TMP planners. For instance, how do generality and flexibility trade off against speed? How does the complexity of writing specifications weigh in? We nevertheless report wall-clock time despite its problems because it remains a widely accepted metric in the community.

Our implementation performs competitively with the state of the art; our evaluation problem is similar to that of Dantam et al. [25] and requires a similar planning time for the same object counts (note that our planning time includes both motion and task planning, whereas the planning time reported by Dantam et al. [25] is only for motion planning).

The average plan size in number of states scales linearly with the number of objects. For the clutter problem, the minimum number of states in a solution plan also scales linearly with the number of objects — if an object is added to the problem with a random position, then in expectation a constant number of states must be added to handle moving that object. This shows that while the plans our implementation produces are not optimal in length, they are only suboptimal by a "constant" factor (technically, by a random variable independent of problem instance size).

Our planning time increases exponentially as object count increases. This is not uncommon in TMP planners solving similar problems, due to the combinatorial nature of the task space. This poor performance may be exacerbated by the particular heuristic we chose for our implementation. This heuristic [51] is naive and has no knowledge of global plan feasibility or motion feasibility. As such, as object count increases, if this heuristic

cannot rank the priorities of the larger set of actions viable in a given state, then our planner will tend to explore the actions broadly rather than following a globally-aware sequence of actions. A more sophisticated heuristic to guide the action search could correct this deficiency. It is also worth noting the large variance in planning times for the larger problem instances. Much of this variance can be attributed to our use of RRT, which has high variance in performance (we do not use random restarts).

We also ran our planner on a variant of the "Kitchen" problem from Lagriffoul et al. [71] (we remove the radishes, which are intended to make manipulation actions more difficult). We can solve the provided instance of this problem in an average of 304.6 seconds over five trials. We were unable to solve the full kitchen problem in reasonable time, necessitating this simplification. We believe that our naive heuristic is unable to give useful priority values for actions in the large task space of the full kitchen problem: by investigating a histogram of the symbolic states our planner visits, we see that it makes significant progress (i.e. it visits symbolic states which accomplish subgoals of the full problem), but explores too broadly — that is, it doesn't commit to a single task plan and see it through to completion, but explores many possible task plans. We believe that using a different heuristic, in particular a heuristic with some notion of global progress toward the goal, would mitigate this problem.

The broad exploration exhibited by our implementation is a potential blessing. By exploring task plans in parallel, the planner avoids overcommitting to a single plan and is robust to task plan branches becoming nonviable without needing to backtrack. For this same reason, we believe that our technique would also be useful in a nondeterminstic execution context, unlike the current deterministic assumption. Upon action failure, we can reuse the planning data structure our technique has grown during the original planning phase to pick up replanning without needing to start over from the initial state.

2.6 Conclusion

This paper contributes a novel, efficient approach to solving integrated task and motion planning problems. Our approach breaks the Gordian knot of TMP by simultaneously searching for a task solution and motion solution in a composite space of symbolic and geometric state. We are able to do so efficiently by using a novel continuous semantics for symbolic predicates describing how "unsatisfied" they are at a given state to search for states which satisfy the preconditions of symbolic actions. Further, our approach is both general and flexible — we can reuse definitions of predicate semantics across domains, search for solutions with different properties by changing predicate semantics, take advantage of action suggestion methods tailored to problem domains without changing our heuristic interface, and make use of any sampling-based planning algorithm simply by using our sampling algorithm.

We present our algorithm and results based on analysis and experimentation with a proof of concept implementation of our sampling algorithm on established benchmarks [71] and show that we perform competitively and can solve realistic problems.

2.6.1 Limitations

Our technique makes a number of assumptions which may limit its applicability. Although our technique could theoretically be expanded to a multi-robot context, we assume the presence of only a single robot. We also do not consider dynamic constraints and assume a quasi-static, deterministic, fully-observable environment. Additionally, our technique makes no independent guarantees of optimality and is reliant on whatever solution optimization the particular planning algorithm using our sampler provides.

2.6.2 Future Work

Our technique has the potential to be embarassingly parallel—multiple instances of our sampler can operate largely independently in parallel, and this parallel potential could be improved by creating independent samplers for each explored symbolic state.

We also hope that our technique's agnosticism to the specific heuristic and sampling-based planning algorithm used will permit investigation of the fundamental relationship between task planning and motion planning to gain insight into techniques for faster, more robust, flexible TMP solving.

CHAPTER 3

COUNTEREXAMPLE-GUIDED REPAIR FOR SYMBOLIC-GEOMETRIC ACTION ABSTRACTIONS

This chapter is adapted from material previously published as:

> Wil Thomason and Hadas Kress-Gazit. *Counterexample-Guided Repair for Symbolic-Geometric Action Abstractions*. May 13, 2021. arXiv: 2105.06537 [cs]. URL: http://arxiv.org/abs/2105.06537

3.1 Introduction

Existing work in TMP has primarily focused on *efficiency* and has produced planners which can solve reasonably complex problems [25, 40, 105, 110, 112]. However, TMP needs more than efficiency to be practically useful. It must also produce *robust* (i.e. successful despite perturbations in control) solutions and be *easy to use* for non-experts. Most current TMP techniques rely on *action abstractions* (models of robot controller dynamics) which are difficult to create correctly and which have implications for planner performance. These abstractions are either bespoke and require expert manual effort to create, or are learned and may be difficult to verify or extend.

As an alternative, we present a technique for automatically constructing and updating interpretable, symbolic action abstractions, called *abstraction repair*. Abstraction repair is the problem of correcting the correspondence between an action abstraction and its underlying physical controller, guided by observations of the controller's execution. Solving this problem makes TMP easier to use because (1) it allows initial (human-provided) abstractions to be simple or even incorrect, without impacting planner success, (2) it can potentially improve planner performance by tailoring an abstraction to its use, and

(3) it can automatically adapt abstractions for different controllers and/or systems with a variety of constraints.

In this work, we (1) adapt *constrained polynomial zonotopes* (CPZs), a computationally efficient non-convex set representation with useful closure properties, as a novel set representation for symbolic-geometric planning, (2) introduce and formally describe the *symbolic-geometric abstraction repair* problem, and (3) contribute an anytime algorithm for symbolic-geometric abstraction repair. The abstraction representation we propose admits an automatically-derived method for sampling states that satisfy arbitrary combinations of constraints. Finding these states is critical to solving TMP problems and usually requires manual effort to implement. Our abstraction representations are also interpretable, reusable, and extensible by humans.

3.2 Related Work

3.2.1 TMP and Action Abstractions

The core of most TMP work is the choice of abstraction, which bridges the symbolic and geometric components of a TMP problem. Though the majority of planners use STRIPS-like action descriptions (i.e. precondition and effect formulae over discrete predicate symbols), the symbolic-geometric abstractions used include bespoke abstraction and refinement function pairs [25], symbolic pose references with action-specific pose samplers [105], several variations on samplers for predicate-satisfying values [27, 40, 54, 110], and others [2, 8, 16, 37, 72]. These approaches require manual specification of the symbolic models of their actions. Toussaint [112], which solves TMP as a continuous optimization problem rather than abstracting geometric state into a symbolic representa-

tion, requires a similar manual specification of the transition constraints between modes corresponding to robot actions. In contrast, we present a hybrid abstraction method based on constrained polynomial zonotopes [66] which permits both efficient sampling of constraint-satisfying states for planning and automatic construction of symbolic models of actions.

Work on learning models of action constraints and effects for TMP has had some recent success [6, 68, 118, 119, 121]. These learning-based approaches tend to create probabilistic models without direct symbolic interpretation [118, 119, 121]; however, interpretability is important for allowing humans to verify and extend action abstraction models. Some of this line of work constructs symbolic abstractions for actions [6, 68], but the symbols used are automatically generated and may be unintuitive for human users. Our work makes a different set of tradeoffs: while our abstractions do not quantify uncertainty, they have intuitive symbolic interpretations, and our chosen set representation provides state samplers for planning for free. Since our approach to creating models of actions is designed around modifying an incorrect starting point as needed, we are able to construct useful abstractions from fewer observations than techniques which are designed to start from no knowledge.

Overall, this paper proposes a middle ground between the interpretable, bespoke representations of traditional TMP planners and newer learned models: a hybrid conservative approximation of action precondition and effect sets which can be symbolically manipulated, and which is interpretable, computationally efficient, and extensible.

3.2.2 Abstraction Repair

The symbolic planning and programming languages literatures also contribute work on plan repair [11, 28, 33, 114] and abstraction refinement [20, 21]. Unlike most of the symbolic planning work on plan repair, our technique to abstraction repair works with hybrid, rather than purely symbolic, state and parameter spaces. The sense in which we mean "abstraction" (a symbolic model of a controller's constraints and effects) also differs from the sense used in most existing formal methods work on abstraction refinement. Some work from the hybrid systems literature [5, 19] is capable of verifying systems with hybrid state, but the goal of this work (to verify that a system satisfies a given safety property) is different from ours, and their use of "abstraction" again differs from ours.

We can view the abstraction repair problem defined in Section 3.3.2 as a model update/repair problem. Though this is a rich area of research [1, 10, 52, 87, 95, 109, 117], we learn a different kind of model than those typically considered. Vemula et al. [117] learns model corrections online for planning with inaccurate models, but its model updates bias the planner away from poorly modeled states rather than improving the correspondence between the model and the modeled controller.

Finally, we draw from the literature on set representations from the formal methods community [3, 4, 66, 67, 78, 93, 101] for the basis of our abstraction representation: the constrained polynomial zonotope (CPZ) [66]. CPZs are well-suited to TMP abstractions because they can express a broad class of non-convex sets, are closed under intersection and union operations, and are computationally and representationally efficient.

3.3 Problem

We introduce the following notation and definitions to formally state the symbolic-geometric abstraction repair problem.

3.3.1 Terminology and Notation

For a given TMP problem instance, let an *object* be a physical entity in the environment, and let a *symbol* be a variable with values taken from an associated domain. The meaning of a symbol is problem-dependent; for example, many manipulation problems use a Boolean symbol to represent whether or not the robot manipulator is empty. We will use $\mathcal{O}$ and $\mathcal{S}$ to denote the sets of all objects and all symbols, respectively. The *state space* for the problem instance is then:

Definition 5: *State Space*

$$\mathcal{Q} = \mathcal{Q}_R \bigtimes_{o \in \mathcal{O}} \mathcal{Q}_o \bigtimes_{s \in \mathcal{S}} \mathcal{D}(s)$$

where $\mathcal{Q}_R$ is the configuration space of a robot R, $\mathcal{Q}_o$ is the configuration space of object $o \in \mathcal{O}$, and $\mathcal{D}(s)$ is the domain of symbol $s \in \mathcal{S}$ (i.e. the set of values s can assume). We assume that $\mathcal{Q}$ is bounded in all dimensions. This state space construction is similar to those of Thomason and Knepper [110] and Vega-Brown and Roy [116].

In a TMP problem, an *action* describes an operation a robot may perform. Typically, and in our use, an action abstracts a parameterized robot controller with a finite execution time (sometimes referred to as a *skill*) by the states from which it can run correctly (the *precondition* or *constraint* of the controller) and the states which are reachable by running the controller (the *effect* of the controller). This is the form of action abstraction used by most existing symbolic planning and TMP work. We define an action accordingly:

Definition 6: *Action* An action α_j abstracts a parameterized controller γ_j and comprises a set of parameter vectors Θ_j, a *constraint function* $\phi_j : \mathcal{Q} \times \Theta_j \to \mathbb{B}$ capturing the conditions under which the action can be used successfully, and an *effect function* $\psi_j : \mathcal{Q} \times \Theta_j \to 2^{\mathcal{Q}}$ describing the possible states of the world after the action completes. α_j is itself a function describing state transformations resulting from the robot executing the controller corresponding to α_j. For $\boldsymbol{q} \in \mathcal{Q}$ and $\theta \in \Theta_j$,

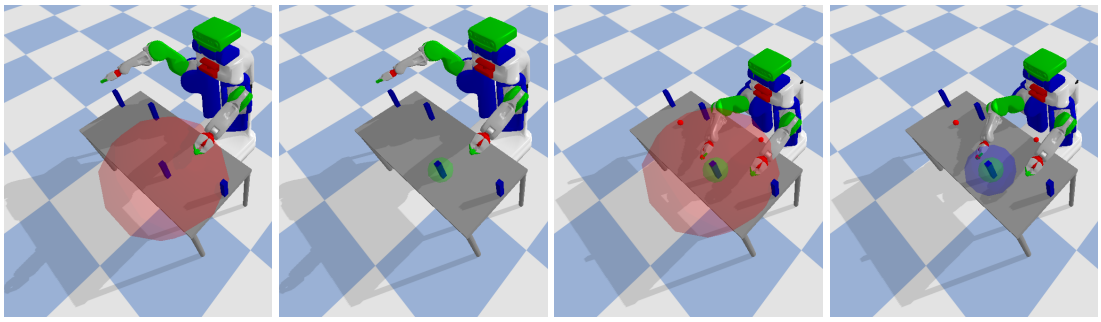
$$\alpha_j(\boldsymbol{q}, \theta) = \begin{cases} \psi_j(\boldsymbol{q}, \theta) & \text{if } \phi_j(\boldsymbol{q}, \theta) = \texttt{true} \\ \{\boldsymbol{q}\} & \text{otherwise} \end{cases}$$

An action α'_j is a *correct abstraction* if its constraint and effect functions model the properties of γ_j exactly. That is, $\forall \boldsymbol{q} \in \mathcal{Q}, \theta \in \Theta_j$, (1) $\phi_\alpha(\boldsymbol{q}, \theta) = \texttt{true} \iff \gamma_j$ runs successfully from $\boldsymbol{q}$ with parameters θ and (2) if $\phi_\alpha(\boldsymbol{q}, \theta) = \texttt{true}$, then $\psi_\alpha(\boldsymbol{q}, \theta)$ is exactly the set of states reachable by running γ_j from $\boldsymbol{q}$ with parameters θ. Similarly, an abstraction is *incorrect* if one or both of these requirements is not met. Intuitively, these conditions mean that α'_j captures the behavior of γ_j accurately—its constraint correctly predicts whether or not γ_j will run successfully, and its effect accurately and precisely models the changes γ_j can cause on the world.

For most realistic actions, it is impossible to verify the above: we cannot enumerate continuous parameters or state components, and some combinations of states and parameters are dangerous for some controllers. As such, we need a notion of correctness with respect to observed controller executions:

Definition 7: *Observations and Correctness* Let an *observation* be a tuple $h_i = (\alpha_j, \boldsymbol{q}, \theta, \boldsymbol{q}')$ recording the result $\boldsymbol{q}'$ of running γ_j from state $\boldsymbol{q}$ with parameters θ. We assume that $\alpha_j(\boldsymbol{q}, \theta) = \{\boldsymbol{q}\}$ if and only if $\phi_j(\boldsymbol{q}, \theta) = \texttt{false}$—that is, if running γ_j does not change the world, then its precondition constraint was not satisfied.

An action α_j is then correct with respect to a set of observations $\mathcal{H}$ if:



(a) The incorrect initial abstraction of the constraint set for the `Pick` action, shown as the red sphere around the blue block.

(b) The true constraint set for the `Pick` action, shown as the green sphere around the blue block.

(c) Counterexample guided abstraction repair samples states (shown as small red spheres) in the flawed abstraction set and checks if the `Pick` action succeeds.

(d) Repair uses sampled counterexamples to update the abstraction to be closer to the (unknown) true constraint set. The updated set is shown as the blue sphere.

Figure 3.1: An example of the abstraction repair process for a simple `Pick` action. The goal of abstraction repair in this instance is to update a flawed initial model of an action's constraint set (Fig. 3.1a) to be closer to the (unknown) true constraint set of the action (Fig. 3.1b), by finding counterexamples: configurations at which the model predicts a different result for the `Pick` action than is observed by running the corresponding controller (Fig. 3.1c), and using these sampled counterexamples to automatically update the model of the constraint set (Fig. 3.1d).

(1) $\forall h_i \in \mathcal{H}$ s.t. $\boldsymbol{q} = \boldsymbol{q}'$: $\phi_j(\boldsymbol{q}, \theta) = \texttt{false}$

(2) $\forall h_i \in \mathcal{H}$ s.t. $\boldsymbol{q} \neq \boldsymbol{q}'$: $\phi_j(\boldsymbol{q}, \theta) = \texttt{true}$ and $\boldsymbol{q}' \in \psi_j(\boldsymbol{q}, \theta)$

Intuitively, Definition 7 means that (1) for all the observations where nothing changed from running γ_j, ϕ_j correctly[1] returned `false`, and (2) for all the observations where γ_j did change the state of the world, ϕ_j correctly returned `true` (indicating that γ_j could be successfully run) and ψ_j correctly included the resulting state.

3.3.2 Symbolic-Geometric Abstraction Repair

We can now define the *symbolic-geometric abstraction repair problem*:

Definition 8: *Abstraction Repair Problem* Given an action α_j such that α_j is an incorrect abstraction of γ_j with respect to a set of observations $\mathcal{H}$, the *symbolic-geometric*

[1] Because γ_j is assumed to not be idempotent.

abstraction repair problem is to construct an action α'_j which correctly abstracts γ_j with respect to $\mathcal{H}$ (per Definition 7).

For a simple example of an abstraction repair problem, consider the following: Let γ_p be a controller for pushing an object on a flat plane to a designated region. γ_p can be executed if (1) the robot manipulator is empty, (2) the robot manipulator is within $5\mathrm{cm}$ of the object to be pushed, and (3) the swept path between the object and the target region is clear. Further let α_p be an action abstracting γ_p. Imagine that ϕ_p, the constraint of α_p is incorrect, and returns `true` whether or not the path between the object and target region is clear (it ignores (3)). The goal of abstraction repair in this instance, then, is to modify ϕ_p to account for (3), given observations of the controller failing when executed with obstacles in the swept path between the target object and target region.

As another example, which we will use as a running example, consider the task of transferring an action abstraction between two robot controllers with different preconditions. For instance, a PR2 may have a grasp controller which can succeed in picking up an object if it is run from a state within $10\mathrm{cm}$ of the target object, regardless of pose. A Baxter may have a similar grasp controller, but, due to its simpler gripper, also require that the manipulator's pose be axis-aligned with the target object. In this instance, the goal of abstraction repair is to—starting from the known action abstraction for the PR2's controller—automatically construct a correct action abstraction for the Baxter's controller, incorporating its stricter requirements.

3.4 Modeling Constraints and Effects

Our approach to abstraction repair uses a new (to TMP) set representation which allows us to represent and manipulate logical formulae over sets of symbolic-geometric states. At a high level, we create *predicate templates* underpinned by symbolic-geometric sets, which allows us to express constraint and effect functions as logical formulae. These formulae conservatively approximate the precondition constraints and effects of robot controllers. Then, upon observing a flaw in an abstraction, we apply an anytime dynamic-programming-based algorithm to "edit" the corresponding formula symbolically. This approach is amenable to using human-provided abstractions as initial input and ensures that repaired abstractions remain interpretable and reusable. In the following sections, we describe our set and predicate representation (Section 3.4.1) and our repair algorithm (Section 3.5) in detail.

3.4.1 Constrained Polynomial Zonotopes

We use *constrained polynomial zonotopes* (CPZs), first introduced by Kochdumper and Althoff [66], to represent sets of symbolic-geometric states (as defined in Definition 5). We briefly overview the structure and properties of CPZs here; the original CPZ paper offers a more detailed treatment [66].

A *zonotope* is the Minkowski sum of a set of generating vectors, and is a commonly used set representation in the formal methods and controls communities due to its efficient representation and closure under operations such as linear map and Minkowski sum. Constrained polynomial zonotopes generalize zonotopes by (1) using polynomial combinations of generating vectors and (2) allowing for polynomial equality constraints

on coefficients. These additions grant CPZs two important properties for our use: the ability to represent many non-convex sets, and closure under intersection and union. CPZs also retain much of the efficient representation and computational efficiency of the simpler zonotopes. More formally (following Kochdumper and Althoff [66]):

Definition 9: *Constrained Polynomial Zonotope* An n-dimensional constrained polynomial zonotope S is a tuple (c, G, E, A, b, R) comprising a starting point $c \in \mathbb{R}^n$, a matrix of generating vectors $G \in \mathbb{R}^{n \times \ell}$, a matrix of exponent vectors $E \in \mathbb{Z}_{\geqslant 0}^{p \times \ell}$, a matrix of constraint generator vectors $A \in \mathbb{R}^{m \times q}$, a vector of constraint equality values $b \in \mathbb{R}^m$, and a matrix of constraint exponent vectors $R \in \mathbb{Z}_{\geqslant 0}^{p \times q}$. This tuple defines the set:

$$S = \left\{ c + \sum_{j=1}^{\ell} \left(\prod_{k=1}^{p} a_k^{E_{(k,j)}} \right) G_{(\cdot,j)} \;\middle|\; \sum_{j=1}^{q} \left(\prod_{k=1}^{p} a_k^{R_{(k,j)}} \right) A_{(\cdot,j)} = b, a_k \in [-1, 1] \right\}$$

CPZs are closed under and have closed-form exact expressions for set intersection and union [66].

Example 1: *Simple CPZ* To build intuition for Definition 9, consider the following CPZ: $S_e = (c_e, G_e, E_e, A_e, b_e, R_e)$, where:

$$c_e = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad G_e = \begin{bmatrix} 2 & 1 & 2 \\ 0 & 0 & 3 \end{bmatrix}$$

$$E_e = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 1 \end{bmatrix} \qquad R_e = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \end{bmatrix}$$

$$A_e = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 7 \end{bmatrix} \qquad b_e = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}$$

This CPZ specifies the set:

$$S_e = \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 \\ 0 \end{bmatrix} a_1 + \begin{bmatrix} 1 \\ 0 \end{bmatrix} a_2^2 + \begin{bmatrix} 2 \\ 3 \end{bmatrix} a_1 a_2 \;\middle|\; \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} a_1 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} a_2 + \begin{bmatrix} 3 \\ 5 \\ 7 \end{bmatrix} a_1^2 a_2^2 = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}, \; a_1, a_2 \in [-1, 1] \right\}$$

3.4.2 Encoding $\mathcal{Q}$ in $\mathbb{R}^n$

We encode a state $\boldsymbol{q} \in \mathcal{Q}$ into a $\boldsymbol{r} \in \mathbb{R}^n$ (where $\mathcal{Q}$ has n dimensions) for compatibility with our CPZ set representation. For $\boldsymbol{q} = (\boldsymbol{q}_R, \boldsymbol{q}_\mathcal{O}, \boldsymbol{q}_\mathcal{S})$, where $\boldsymbol{q}_R$ is the robot configuration vector, $\boldsymbol{q}_\mathcal{O}$ is the vector of object configuration vectors, and $\boldsymbol{q}_\mathcal{S}$ is the vector of symbol values, $\boldsymbol{r} = (\boldsymbol{q}_R, \boldsymbol{q}_\mathcal{O}, \texttt{enumerate}(\boldsymbol{q}_\mathcal{S}))$. $\texttt{enumerate}(\cdot)$ is a function mapping each value in the domain for each symbol to an integer. For example, a Boolean symbol with domain $\{\texttt{true}, \texttt{false}\}$ may be mapped as follows: $\{\texttt{true} = 1, \texttt{false} = 0\}$. The inverse of $\texttt{enumerate}(\cdot)$ takes a value $r_s \in \mathbb{R}$ to a value $v \in \mathcal{D}(s)$ by treating the values assigned by $\texttt{enumerate}(\cdot)$ as endpoints of intervals of real numbers and determining the bin containing r_s. This encoding may result in a CPZ containing points which are not valid states (e.g. not all vectors of real numbers will correspond to a valid object pose in SE(3)); this shortcoming is acceptable for our use, as we are conservatively approximating sets of valid states.

3.4.3 Modeling Constraints and Effects

We model constraint and effect functions as formulae over CPZs, with intersection and union taking the place of logical "and" and "or", respectively. We assume that logical negation is implemented at the atomic (set) level. This representation matches the symbolic formulae used in most existing TMP work and serves as a computationally efficient conservative approximation for arbitrary constraint and effect functions. Although this representation is not as general as allowing constraint and effect formulae to be arbitrary functions, it matches the representation used in most existing TMP work and suffices to represent constraints and effects for many realistic problems. A constraint function ϕ_j, then, returns true for $\phi_j(\boldsymbol{q}, \theta)$ if and only if $\boldsymbol{q}$ is contained in the CPZ described by the formula for ϕ_j with parameters θ. Similarly, an effect function ψ_j returns the CPZ described by the formula for ψ_j from state $\boldsymbol{q}$ and with parameters θ.

3.4.4 Predicate Templates

Although arbitrary CPZs can be used for these formulae, we restrict the formulae used in our repair process to a set of CPZs described by *predicate templates*. A *predicate*, for our use, is a Boolean function defining a property of a state (e.g. a predicate might be `true` if a robot's manipulator is within some given distance of a particular object in a state, and `false` otherwise—describing the property of being "at" the object). A predicate template describes a parameterized predicate, and concisely describes a whole class of properties.

Definition 10: *Predicate Template* A predicate template P_i pairs a *constraint space transform* with a template for generating a CPZ. A constraint space transform is a function $g_i : \mathcal{Q} \rightarrow \mathbb{T}_i$ (for a predicate-specific bounded $\mathbb{T}_i \subset \mathbb{R}^{m_i}$, $m_i \in \mathbb{Z}_{>0}$), which maps states

to a predicate-specific constraint space—a subspace of the real numbers in which the associated CPZ will be created.

Constraint space transforms are related to the nonlinear constraint state maps of Chou, Ozay, and Berenson [18], and allow us to apply arbitrary nonlinear transformations to states before describing sets as CPZs to represent a desired property. The rationale for using constraint space transforms is that it is often easier to describe a given state property in a transformed state space—for example, as shown in Example 3, it is straightforward to create a CPZ expressing a distance constraint between a robot link and an object in the workspace, but describing this same property in the un-transformed configuration space is challenging. Further, we use both constraint space transforms and CPZs (rather than just having constraint space transforms describe predicates themselves) because of the compositional properties of CPZs. Common constraint space transforms include set-theoretic projection (for a predicate which tests only a subset of the state) and forward kinematics (for predicates best expressed in the robot's workspace).

A template for generating a CPZ is a function T_i from a predicate-specific parameter space Θ_{P_i} to the space of parameterized m_i-dimensional CPZs. T_i constructs a parameterized CPZ representing the set of points in $\mathbb{R}^{m_i}$ (and therefore the set of states in $\mathcal{Q}$) that satisfy the property described by P_i. The remaining unbound parameters in the result of T_i are bound to values from the parameter vector of the constraint/effect function in which the result is used. For instance, in Example 3, the unbound parameter `obj.center` in the template result is bound to the value of the `obj` parameter for the action in which the distance predicate is used.

Restricting constraint and effect functions to formulae over CPZs generated from a known set of predicate templates allows us to maintain interpretability when repairing an abstraction and provides useful structure to the repair search problem. Further, many

predicate templates are applicable across a range of problems and environments, making a small set of "basis" predicates sufficient to describe a rich set of conditions.

For example, we can define a predicate template to test for the value of a Boolean symbol as follows:

Example 2: *Symbol Value Predicate Template* Let s be a Boolean symbol in the state space $\mathcal{Q}$ of a problem. Then the constraint space transform is $g(\boldsymbol{q}) := \texttt{proj}_s(\boldsymbol{q})$, where $\texttt{proj}_s(\cdot)$ projects a state $\boldsymbol{q}$ to its value for s. The CPZ template for this predicate is the degenerate CPZ $([1], [], [], [], [], [])$. Since we encode Boolean values as either 0 or 1, the above CPZ only contains states in which $s = \texttt{true}$.

Similarly, a predicate template testing the distance between a robot link and an object in the environment looks like:

Example 3: *Distance Predicate Template* This predicate template is parameterized by a distance d. The constraint space transform is the forward kinematics function $g(\boldsymbol{q}) := \texttt{FK}(\boldsymbol{q})$ for the target link. The CPZ template for this predicate is then $(c_{P_d}, G_{P_d}, E_{P_d}, A_{P_d}, b_{P_d}, R_{P_d})$ with

$$c_{P_d} = \texttt{obj.center} \qquad b_{P_d} = 0.5$$

$$G_{P_d} = dI_3 \qquad E_{P_d} = \begin{bmatrix} I_3 \\ \mathbf{0} \end{bmatrix}$$

$$A_{P_d} = \begin{bmatrix} 1 & 1 & 1 & -0.5 \end{bmatrix} \qquad R_{P_d} = \begin{bmatrix} 2I_3 & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$$

where I_n is the $n \times n$ identity matrix. This CPZ defines the ball of radius d around the object $\texttt{obj}$, which is bound to a parameter of the action in which this predicate template is used. For an object centered at (x, y, z) and a particular distance value $d \in \mathbb{R}$, this

produces:

$$\texttt{Distance}(\texttt{obj}, d) = \left(\begin{bmatrix} x \\ y \\ z \end{bmatrix}, \begin{bmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & d \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 & -0.5 \end{bmatrix}, \begin{bmatrix} 0.5 \end{bmatrix}, \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right)$$

3.4.5 Combining CPZs with different constraint transforms

If two CPZs resulting from different predicate templates have different constraint transforms, then they must be converted to lie in a common space before we can apply intersection and/or union operations.

The intuition behind this common space conversion is that a CPZ defined for a set of dimensions $\{d_1, \ldots, d_n\}$ does not constrain any other dimensions. Thus, given that each dimension of $\mathcal{Q}$ is bounded and that the range of each constraint space transform is bounded (from Definitions 5 and 10, and which implies that any transformed dimension is also bounded) , we can re-express any pair of CPZs in a common space by adding additional generators spanning, for each original CPZ, the dimensions constrained only by the other. The common space CPZs are each such that (1) when projected down to its original dimensions, it is equivalent to the corresponding original CPZ, and (2) when projected down to the dimensions it did not constrain, it includes all points in those dimensions.

We now present the full algorithm for constructing such a common space and representing each CPZ in it.

Let $S_1 = (c_1, G_1, E_1, A_1, b_1, R_1)$ lie in the space comprised of dimensions $D_1 = \{d_{1,1}, \ldots, d_{n,1}\}$, and let $S_2 = (c_2, G_2, E_2, A_2, b_2, R_2)$, lying in the space comprised of dimensions $D_2 = \{d_{1,2}, \ldots, d_{m,2}\}$. We construct the common space representations S_i^c as follows. First, we must re-order the dimensions of S_1 and S_2 to be consistent. The common space for S_1 and S_2 is $D_c = D_1 \cup D_2$. We construct the ordered set $D' = \{D_1 \backslash D_2\} \cup \{D_1 \cap D_2\} \cup \{D_2 \backslash D_1\}$ and rewrite c_i and G_i to c_i' and G_i' (respectively) by re-ordering their rows to match the order of dimensions defined by D' for the dimensions in $D_i \cap D'$ (i.e. the dimensions constrained by S_i).

Next, we form the matrix of generators for the dimensions unconstrained by S_i by constructing

$$B_{D' \backslash D_i} = \begin{bmatrix} \frac{e_1}{2} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \frac{e_\ell}{2} \end{bmatrix}$$

where e_j is the extent of each dimension in $D' \backslash D_i$, in the order defined by D'. Finally, let m_j be the middle point of each dimension in $D' \backslash D_i$, also in the order defined by D'.

We then have:

$$S_1^c = \left(\begin{bmatrix} c_1' \\ m_1 \\ \vdots \\ m_\ell \end{bmatrix}, \begin{bmatrix} \mathbf{0} & G_1' \\ B_{D' \setminus D_1} & \mathbf{0} \end{bmatrix}, \begin{bmatrix} \mathbf{0} & E_1 \\ I_\ell & \mathbf{0} \end{bmatrix}, A_1, b_1, \begin{bmatrix} R_1 \\ \mathbf{0} \end{bmatrix} \right)$$

$$S_2^c = \left(\begin{bmatrix} m_1 \\ \vdots \\ m_\ell \\ c_2' \end{bmatrix}, \begin{bmatrix} B_{D' \setminus D_2} & \mathbf{0} \\ \mathbf{0} & G_2' \end{bmatrix}, \begin{bmatrix} I_\ell & \mathbf{0} \\ \mathbf{0} & E_2 \end{bmatrix}, A_2, b_2, \begin{bmatrix} \mathbf{0} \\ R_2 \end{bmatrix} \right)$$

where I_ℓ is the $\ell \times \ell$ identity matrix and $\mathbf{0}$ is an appropriately-sized block of zeros. Intuitively, by adding the m_j to each center and the generators with $\frac{e_j}{2}$ for each missing dimension, we have added spanning vectors for the missing dimensions (because, for CPZs, the coefficients a_i range from -1 to 1). This corresponds to the intuition above that any dimensions not present in the original CPZ should be (effectively) unconstrained in the common space representation of the CPZ. We can then take the intersection or union (respectively) of S_1^c and S_2^c as usual.

In most practical cases, CPZs representing predicates exist in one of a few shared constraint spaces (e.g. pose space or a space containing a single symbol), and thus this common space transformation is often not needed.

3.5 Abstraction Repair Algorithm

Our approach to abstraction repair is an anytime algorithm based on symbolic "edits" to a constraint or effect formula. Intuitively, because we represent each constraint or effect as a finite-length logical formula over symbolic predicates backed by CPZs, any incorrect

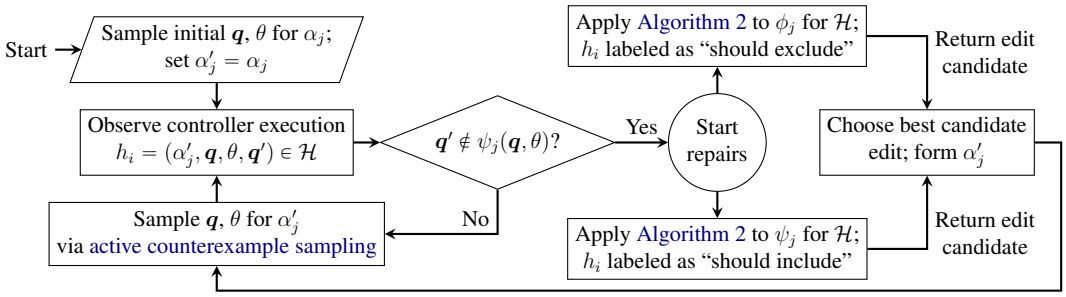


Figure 3.2: Our abstraction repair algorithm begins by choosing an action α_j and sampling a state $\boldsymbol{q}$ and parameter values θ. We execute the controller $\gamma_j(\boldsymbol{q}, \theta)$ and observe the resulting state $\boldsymbol{q}'$. If $\boldsymbol{q}'$ is "expected" (i.e. already contained in the set described by $\psi_j(\boldsymbol{q}, \theta)$) we draw a new sample according to our active counterexample sampling algorithm. If $\boldsymbol{q}'$ is "unexpected" (i.e. not contained in $\psi_j(\boldsymbol{q}, \theta)$) we run Algorithm 2 for both possible repairs (either a flaw in the constraint or in the effect for α_j) until either the timeout is reached or the searches terminate. We then choose the candidate edit with the lowest error, update the action abstraction accordingly, and draw a new sample according to our active counterexample sampling algorithm.

formula can be transformed into the correct formula for its constraint or effect by applying a finite series of edit operations which either add, remove, or replace predicates, or which modify the parameters of a predicate template. Further, we observe the following sound (but incomplete[2]) recurrence relation:

$$\texttt{best}(f_{i+1}) = \min\Big\{ \min_{\texttt{add}}(\texttt{best}(f_i)),\ \min_{\texttt{remove}}(\texttt{best}(f_i)),\ \min_{\texttt{replace}}(\texttt{best}(f_i)),\ \min_{\texttt{param}}(\texttt{best}(f_i)) \Big\}$$

where f_i is the i^{th} edit applied to formula f, $\texttt{best}(f_0) = f_0$, and $\min_{\texttt{op}}(\cdot)$ computes the lowest-error single edit using $\texttt{op}$ as its operation. There are several feasible candidates for the error function used in the $\min$ computations above. Most directly, we can measure the distance to the set boundary of observations with the wrong set inclusion value (i.e. if an observation should have been included in the set described by a formula but is not, or vice versa). This metric intuitively describes the correctness of a formula as defined in Definition 7 as well as the "amount" of incorrectness for misclassified observations. We could also favor conservative edits to a formula by using change in set measure as a

[2]This recurrence is incomplete for optimally solving abstraction repair because an intermediate edit may have a higher error and thus not be chosen.

component of the error function.

We show the repair process loop in Fig. 3.2. This process iteratively samples states and parameter values from which to attempt a given controller γ_j abstracted by an action α_j, and notes whether or not the result of running γ_j is unexpected. If it is, the process attempts to repair both ϕ_j and ψ_j. We perform repairs on both formulae because, without additional knowledge about the *intended effect* of γ_j, it may be impossible to distinguish between an unexpected result caused by an underconstrained constraint formula and an unexpected result caused by an overconstrained effect formula. The process in Fig. 3.2 is one possible process for repair using Algorithm 2—the core abstraction repair algorithm can also be used in the process of executing a plan, or in an interactive mode of operation with human guidance.

We provide pseudocode for the main edit search algorithm in Algorithm 2, which implements an anytime version of the above recurrence relation. In Line 1, we extract the improperly classified observations from the set $\mathcal{H}$. Then, in Lines 2 to 4, we find the initial error value for the flawed formula f and compute the initial set of candidate edits to the formula. We then iterate through the set of edits until either time runs out or we have evaluated every candidate edit. For each candidate edit, we first evaluate the edit operation to get the resulting formula (Line 9), then compute its error with respect to the set of observations (Line 10). If the candidate's error is lower than or equal to the current best error, we find the new set of incorrectly classified observations and add new candidate edits to the queue (Lines 11 and 12). If the candidate has lower error than the current best formula, we also update the tracked best in Lines 13 to 15. Finally, when we're out of edits or time, we return the current best formula.

Algorithm 2 relies on a few auxiliary functions: `Error` computes the error of a formula relative to a set of observations, and can be any of a variety of error functions (as

discussed above). `Unexpected` is a Boolean function that computes whether or not an observation is correctly classified by a formula; `filter` is the standard filter function, which (in our use) returns the set of observations for which `Unexpected` returns `true`. `time` returns the elapsed time since the algorithm started running.

The `GenerateEdits` function computes the set of relevant edits for a formula and set of observations. It iterates through each observation in the set and finds the set of edits adding a predicate, replacing a predicate, removing a predicate, and re-optimizing the parameters of a predicate. Each of these four edit operations can be implemented naively, by enumerating the possible predicates, or can use information about the formula f and observation h to filter the set of edits returned (e.g. by analyzing if the observed states are correctly classified in any subset of their dimensions, and only returning predicates which apply to the remaining dimensions). Parameter optimization can be implemented differently for different predicates, but generally involves solving a nonlinear program minimizing `Error` over a predicate's continuous parameters, nested in an enumerative search over the domains of the predicate's discrete parameters. We optimize the parameters of a single predicate at a time (i.e. we do not explicitly solve joint parameter optimization problems).

We additionally require that the formula f is expressed in disjunctive normal form (i.e. a disjunctive combination of conjunctive clauses) to simplify the search space. This restriction reduces the number of possible sites in the formula at which to add a predicate (one site per conjunctive clause to add each predicate with logical "and", and one site to add each predicate as its own clause with logical "or" at the top level of the formula).

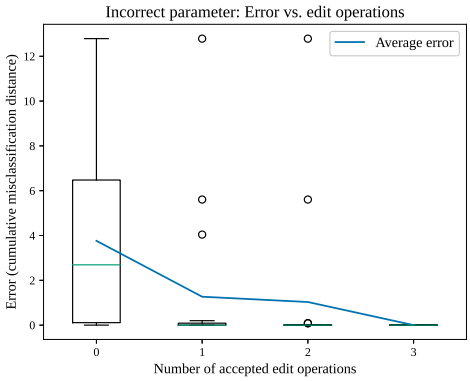
Algorithm 2: Anytime Abstraction Repair

Input: Formula f; observation set $\mathcal{H}$; timeout t
Output: Formula f' with lower error w.r.t. $\mathcal{H}$

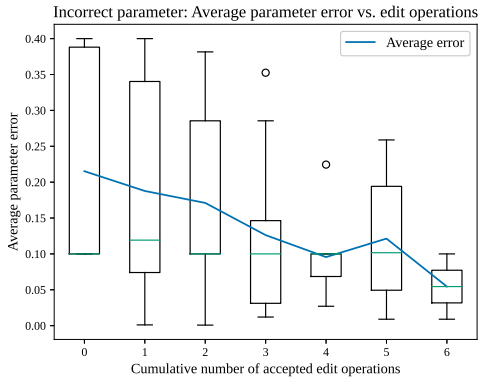
```
1  H_I ← filter(Unexpected(f), H)
2  err ← Error(f, H)
3  best ← f
4  GenerateEdits(f, H_I, edits)
5  while time() ⩽ t do
6      if empty(edits) then return best
7      /* Lazily evaluate next edit                    */
8      e ← NextEdit(edits)
9      f' ← eval(e)
10     ε ← Error(f', H)
11     if ε ⩽ err then
12         GenerateEdits(f', H_I, edits)
13         if ε < err then
14             err ← ε
15             best ← f'
16 return best
```

3.5.1 Active Counterexample Sampling

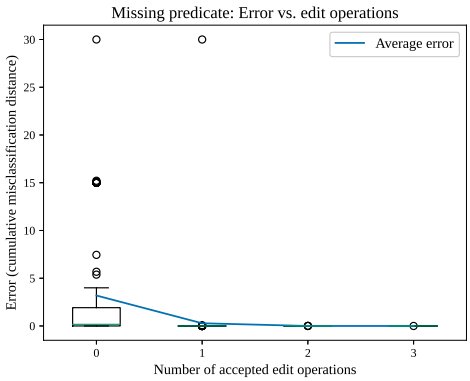
The performance of abstraction repair depends on the observations it receives. While Algorithm 2 will never make a formula less correct with respect to a set of observations, purely random observation sampling may not discover instances of error in a formula. To mitigate this issue, we introduce a form of active sampling, which heuristically samples start states and parameter values more likely to unearth a counterexample for a given formula. This active sampling algorithm is straightforward: after completing a repair for an action abstraction, we sample from states that satisfy the old constraint formula ϕ for a state which does *not* satisfy the new constraint formula ϕ' (or vice versa). The intuition for this heuristic is that we are more likely (compared to naive sampling) to find erroneously included or excluded start states by sampling in the difference between the old and new constraint sets; this space captures the set of states which may have been missed due to over-generalization from previous samples. For completeness, we also



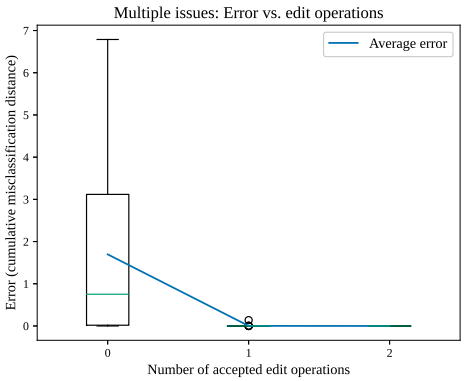
(a) Classification error over time within a repair loop for experiment 1.



(b) Parameter value error over a sequence of repair instances for experiment 1.



(c) Classification error over time within a repair loop for experiment 2.



(d) Classification error over time within a repair loop for experiment 3.

Figure 3.3: Figs. 3.3a, 3.3c and 3.3d show that, within one repair loop, subsequent edit operations lead to decreased error on average. There is relatively high variance in initial error (i.e. when zero edits have been applied but the error has been calculated), as well as several significant outliers (circles) in overall error. The outliers in later steps are explained by the predicate parameter optimization sub-procedure encountering a local optimum or otherwise failing. However, within at most three edit operations, all instances are able to achieve roughly zero error over their sub-sampled set of observations. Fig. 3.3b shows that, for experiment 1, the value of the initially incorrect constraint parameter approaches the correct value as additional unexpected observations are made and repairs are performed. As before, the non-monotonic variation in the parameter error is due to local optima and observation sub-sampling.

return naive samples (i.e. sampled directly from ϕ' without considering ϕ) with nonzero probability.

3.6 Evaluation

We provide and evaluate an open-source Python proof-of-concept implementation of abstraction repair[3] on several pick-and-place-inspired repair operations. We show that we can repair, in a low number of unexpected observations[4], realistic errors in constraint formulae[5]. Additionally, we demonstrate that abstraction repair monotonically improves the quality of abstractions (with respect to a set of observations), making it suitable for anytime use.

All of the following experiments are single-threaded and run on a 3.7 GHz AMD Ryzen 7 2700X CPU, with 32 GB RAM. We use an error function computing the squared minimum distance from an observation state to the boundary of a formula's CPZ. This error function is smooth but not necessarily convex; we choose it because it captures the "classification error" of a formula (i.e. "how much" it incorrectly excludes or includes an observation). Given (1) that this error function is computationally expensive (each point-CPZ distance computation requires solving a nonlinear program) and (2) that we want to evaluate the use of abstraction repair in an anytime context, we limit each invocation of the repair loop to a total of 100s of computation time. Further, we *sub-sample* observations for each invocation of the repair loop. Given a set of unexpected observations U at the time we invoke the repair loop, we select a random subset of the current expected observations E such that $|U| = |E|$. This sub-sampling is important (1) for computational feasibility, as computing the error function on the full set of observations is expensive, but also (2) to balance the expected and unexpected observations, and

[3] Link to Github repository omitted for double-blind review.

[4] Observations for which the prediction of the abstraction under repair did not match the result of executing the corresponding controller

[5] We only evaluate on constraint formula repairs; the repair process for effect formulae only changes which state in an observation is considered.

avoid failure cases where error is minimized by forcing a formula's CPZ to be empty (e.g. if most examples should be excluded). However, sub-sampling does have a cost: whereas Algorithm 2 monotonically improves a formula given access to the full set of observations, sub-sampling can introduce variation in the error resulting from repair, leading to sub-optimal parameter values and solutions. We address improvements to sub-sampling in future work. Finally, we run repeated trials of each experiment in simulation, randomly initializing the placement of objects each time, and randomly sampling states. Figs. 3.3a to 3.3d summarize the results of the following experiments; in these figures, outliers are shown as circles, the mean error value at each point is shown as a trend line, and the distribution of data at each point is shown as a vertical box plot. For Figs. 3.3a, 3.3c and 3.3d, the y-axis shows error in terms of our aforementioned "misclassification" error function. For Fig. 3.3b, the y-axis shows the error between the actual and target value of the continuous parameter in the formula under repair.

3.6.1 Incorrect predicate parameter

As a simple example, we start by showing a repair for a single-predicate constraint formula with an incorrect continuous parameter value. Consider a constraint for a basic `Pick` action requiring the manipulator to be within a certain distance of the target object. The correct version of this formula is the single predicate $(\texttt{dist}\ obj\ manip\ 0.1)$, which says that the object obj and manipulator $manip$ must be within 10cm for the grasp controller underlying the `Pick` action to successfully run. We start with an incorrect version of the constraint formula, $(\texttt{dist}\ obj\ manip\ 0.5)$, which is correct except for the continuous parameter value. Our goal, then, is to discover the correct parameter value via a series of optimization repairs in response to unexpected observations. We ran ten trials of this repair problem, using naive (i.e. non-active, uniform random) sampling

for states from which to attempt the `Pick` action. Each trial runs until it finds five unexpected observations (and therefore invokes the repair loop five times). Fig. 3.3a shows that, averaged over trials and invocations of the repair loop, we are able to decrease the formula error evaluated on the current set of observations to zero within three edit operations per loop. Further, Fig. 3.3b shows the error in the continuous parameter value over the number of edits applied[6], averaged over trials and cumulative over invocations of the repair loop. This trend shows that we do converge toward the correct value of the continuous parameter, although observation sub-sampling does cause the convergence to be non-monotonic.

3.6.2 Missing constraint predicate

Next, we demonstrate that we can repair the earlier running example of transferring an action abstraction between two robot controllers with different constraints. We start with a constraint formula $(\texttt{dist}\ obj\ manip\ 0.1)$, which may be sufficient for a robot with a more sophisticated `Pick` controller that can align the manipulator with the object to be grasped. Our goal is to discover that for another robot with a simpler `Pick` controller, this formula is missing a predicate constraining the manipulator's orientation. Concretely, we wish to find the formula $(\texttt{and}\ (\texttt{dist}\ obj\ manip\ 0.1)\ (\texttt{roll}\ obj\ manip\ 0.1))$, where the `roll` predicate constrains the roll of the manipulator $manip$ to be within 0.1 radians of the roll of the object obj in the global frame. In this case, uniform random sampling is too inefficient to be reasonable. The target formula represents a set of small measure in configuration space, so the probability of naively sampling a counterexample is similarly small and doing so takes (in expectation) a large number of wasted samples. Instead, we

[6]Note that each unexpected observation causes an invocation of the repair loop, but each invocation of the repair loop may apply multiple edits.

employ the active counterexample sampling algorithm described in Section 3.5.1, and sample directly from the repair formula's CPZ. We ran 20 trials of this repair problem, terminating each trial after ten unexpected observations (invoking the repair loop a total of ten times per trial). Fig. 3.3c shows that, as with experiment 1, the error (again, averaged over all trials and repair loop invocations) goes to near-zero within one to three edit operations. However, there is an important caveat to this result, and to this approach to abstraction repair in general: because our notion of error is defined purely by the chosen error function, it is possible for a formula a human would call spuriously correct to achieve as low an error as the correct formula, and abstraction repair has no way to distinguish which is "better". More concretely, although in some trials we discover formulae such as $(\texttt{and}\ (\texttt{dist}\ \mathit{obj}\ \mathit{manip}\ 0.1)\ (\texttt{roll}\ \mathit{obj}\ \mathit{manip}\ 0.49))$ and $(\texttt{and}\ (\texttt{dist}\ \mathit{obj}\ \mathit{manip}\ 0.08)\ (\texttt{roll}\ \mathit{obj}\ \mathit{manip}\ 0.02))$, which are reasonably close in structure and parameters to the correct formula, we also discover in other trials formulae such as $(\texttt{empty}\ \mathit{manip})$ and

$$(\texttt{and}\ (\texttt{dist}\ \mathit{obj}\ \mathit{manip}\ 4.1)\ (\texttt{roll}\ \mathit{obj}\ \mathit{manip}\ 1.3)\ (\texttt{empty}\ \mathit{manip}))$$

which are clearly only spuriously correct for a particular set or sub-sample of observations (the `empty` predicate requires that the manipulator manip is empty, which we model with a symbolic state component). Resolving this problem (and improving the performance of the search over edits) requires incorporating human knowledge (e.g. by presenting repair candidates with equivalent error to a human for review) or a more sophisticated selection heuristic. We consider this issue a primary focus for future work.

3.6.3 Multiple constraint errors

Finally, as both of the preceding examples have required only a single change (in the optimal case); we now show that we can repair formulae with multiple problems. We want to find the formula:

$$(\texttt{and}\ (\texttt{distance}\ obj\ manip\ 0.1)$$
$$(\texttt{roll}\ obj\ manip\ 0.1)$$
$$(\texttt{empty}\ manip))$$

starting from the formula $(\texttt{distance}\ obj\ manip\ 0.7)$, which is missing both a continuous and a discrete predicate and has an incorrect parameter value. As before, we ran 20 trials, stopping either after reaching 100 unexpected observations or making 1000 uninterrupted expected observations (indicating convergence). Fig. 3.3d shows that we are able to find a formula with approximately zero error within at most two edit operations (averaged across trials and repair loop invocations); as before, we sometimes choose repaired formulae close to the desired target, and sometimes choose formulae which are only spuriously low-error.

3.7 Conclusions and Future Work

Integrated Task and Motion Planning, or TMP, has potential as a means of granting robots expanded autonomy, but current TMP techniques require substantial expert manual effort to use, primarily in specifying symbolic action models and samplers for constraint-satisfying states. The symbolic-geometric abstraction repair we propose in this work mitigates this shortcoming by allowing action models to be initially incomplete or incorrect, and therefore easier to specify. Coupled with our proposed novel predicate

representation, based on constrained polynomial zonotopes [66], abstraction repair can automatically produce both symbolic action models and samplers for constraint-satisfying states. We additionally contribute an anytime algorithm for performing abstraction repair, which monotonically improves the correctness of an action abstraction with respect to a set of observations.

In future work, we will extend our proposed abstraction representation to (1) relax the need for a known set of predicate templates, (2) add logical negation to our CPZ logic, and (3) express quantified uncertainty about action constraints and effects, to better model probabilistic actions. We will additionally improve on our algorithm for abstraction repair with regards to generalizing observations and distinguishing between solution candidates with equivalent error. Generalizing individual observations will improve the observation efficiency of our approach to abstraction repair, reduce the need to store a large set of observations to compute abstraction error, and permit us to more efficiently create constraint and effect formulae with universal and existential quantifiers. Distinguishing between edit candidates with equivalent error is a challenging problem, but could involve active learning with non-expert human users answering questions about possible repairs, or a learned heuristic approximating repair preference.

Beyond improving upon abstraction repair, we intend to explore improvements to TMP enabled by abstraction repair and our CPZ-based predicate representation. Abstraction repair opens the door to online plan repair: minimally altering a plan made under a flawed abstraction to create a correct plan under the repaired abstraction. This would allow robots to adapt to unexpected action failures efficiently, on the fly. Further, CPZ-based predicate representations offer interesting possibilities for TMP planning, including estimating action difficulty by approximating precondition set measure and rapid testing for geometric feasibility via set intersection.

CHAPTER 4

EFFICIENT OPTIMAL SAMPLING-BASED INTEGRATED TASK AND MOTION PLANNING VIA JUST-IN-TIME ACTION GROUNDING AND AS-NEEDED GEOMETRIC SYMBOLIFICATION

This chapter is based on joint work with Jonathan D. Gammell and Marlin P. Strub.

4.1 Introduction

This chapter presents an efficient optimal sampling-based algorithm for integrated task and motion planning (TMP) based on two key insights: (1) lazy, motion-planner-guided selection of continuous action parameters to avoid wasted work, and (2) a novel form of optimistic symbolic-geometric planning which can ignore continuous preconditions until they become necessary.

As earlier chapters have discussed, TMP is a holistic approach to solving robot planning problems with complex, interacting symbolic and geometric constraints, specified by high-level goal constraints. Useful TMP is necessarily efficient (among other properties); humans do not want to stand around waiting while their robot deliberates how to complete a simple task. As such, a key challenge for efficient TMP is determining how to spend computational effort while solving a problem. The space of candidate symbolic plans is intractably large for non-toy problems, and—as noted by Dantam et al. [25] in motivating their constraint-based task planning approach for TMP—unlike in "standard" symbolic planning, TMP usually needs to explore multiple such candidates. Each candidate plan requires significant computational effort to evaluate: the planner must choose continuous parameters for actions that result in physically feasible motion subgoals and do not preclude later steps in the plan (subproblem (2)) and find a complete motion plan

(subproblem (4), itself computationally hard, involving collision checking of potentially many states, many nearest-neighbor queries, etc. in the sampling-based motion planning paradigm). Further complicating matters is the fact that most practically usable motion planners (i.e., modern sampling-based and trajectory optimization approaches) cannot decide plan nonexistence. Thus, we are left needing to know, as early as possible, when a candidate symbolic plan is infeasible (and ideally *why* it is infeasible, to guide selection of a modified task plan), but potentially unable to ever know with certainty that this is so.

Given this state of affairs, we must prioritize avoiding motion planning and parameter search computation—more specifically, avoiding *wasted* motion planning and parameter search computation. We also ought to prioritize methods of pruning the search space of candidate symbolic plans as aggressively as we can without losing possible solutions, to further reduce the amount of computation we need before finding a feasible candidate. One source of infeasible symbolic plan candidates stems from "hidden" geometric constraints: geometric preconditions for an action which are not included in an action's symbolic model. Although this may be the result of a flawed symbolic specification, as discussed in Chapter 3, it can also arise for good reason. Consider the example of retrieving a set of objects from within a cluttered cupboard: a reasonable `pick` action for this problem domain might specify that the robot manipulator is within some distance threshold of the target object, and that the manipulator is empty. It might not, however, specify that the target object must be unobstructed by all other objects, as this requires either prediscretization (to create an occupancy grid in the symbolic state space) or expensive computation (e.g., computing a swept-volume for the manipulator) to decide whether an object is unobstructed when attempting to find parameters for the action (e.g., the grasp pose). Further, including these additional constraints makes it more challenging to find any candidate task plan, regardless of how the constraints are implemented. Altogether, we want a method of pruning potential symbolic plan candidates based on

these geometric constraints without having to pay the cost of including them in every iteration of symbolic planning.

With this in mind, we present an algorithm for integrated task and motion planning that combines a batch-sampling-based motion planner with a differentiable distance function representation for geometric predicates to defer sampling of action parameters until they are necessary for motion planning, and maximally reuse motion planning effort across alternative candidate symbolic plans. This algorithm further permits symbolic planning in a relaxed domain that ignores geometric constraints until it has gathered evidence that they are necessary for a given step of the symbolic plan—benefitting from the ability of geometric constraint symbols to prune the space of candidate symbolic plans without having to pay the majority of their cost. Finally, the sampling-based planner underpinning our algorithm efficiently computes asymptotically optimal motion plans, allowing us to extend this ability to find asymptotically optimal (in motion cost) task and motion plans.

4.2 Related Work

This planner builds upon and combines ideas in several prior planners from the motion planning and TMP literature. Gammell, Srinivasa, and Barfoot [34] originated the core approach of batch-sampling-based motion planning that we extend in this paper. The insight of this family of methods is that, by drawing batches of valid samples in a planning state space, one builds an implicit random geometric graph (RGG) encoding an approximation of the free state space that improves in accuracy with every subsequent batch. This strategy allows a planner to efficiently evaluate states for inclusion in a solution path in order of their estimated solution quality, accelerating optimal motion

planning by reducing the number of states which must be processed. Strub and Gammell [106] build on this idea by updating the heuristic used to order state evaluation via an efficient reverse search from the goal to the start state. Importantly for our use, the reverse search checks state validity but defers evaluating expensive edge validation until it has more information about the set of edges likely to be part of a solution.

Our approach to symbolic planning is closely related to the incremental SAT-based method proposed by Dantam et al. [25], which itself closely adapts the SATPLAN algorithm originated by Kautz, Selman, et al. [59]. In contrast to Dantam et al. [25], however, we neither include a representation of the scenegraph in our discrete state nor prediscretize the continuous state, do not require a translation step between discrete and continuous state, and form additional constraints to request alternative symbolic plans from the task planner differently.

Finally, we adopt the composite state space and continuous predicate representation of Thomason and Knepper [110] (i.e., Chapter 2). This predicate representation forms the basis of our task-motion interface, as it allows us to directly sample states satisfying action preconditions formulae and guide the motion planner toward these states.

Separately, Srivastava et al. [105] also includes a mechanism for adding symbolic constraints to the task planning problem based on reasons for motion planner failure. Their approach, however, assumes that continuous variables are represented by symbolic references to finite sets of values. Many other works perform multimodal motion planning for manipulation [7, 8, 16]. Some others offer almost-surely asymptotically optimal manipulation planning [99], but only work for manipulation planning problems and assume the presence of domain-specific samplers for mode transitions. Schmitt et al. [99] in particular requires an offline phase to precompute a roadmap for the manipulation problem, whereas we operate without any precomputation. Shome, Nakhimovich, and

Bekris [104] show that asymptotic optimality for integrated task and motion planning under realistic assumptions is not affected by the existence of singular point transitions between modes, i.e., actions which may only be taken from a single state. Recent work in integrated task and motion planning [98] takes an alternative approach to compensating for incomplete symbolic planning specifications: rather than adding symbolic constraints, they compute the top-k plan candidates and use Monte Carlo tree search to decide how to take "detours" between plans when a candidate is determined to be infeasible. Finally, although the two approaches are fundamentally different, Garrett, Lozano-Pérez, and Kaelbling [39] also introduce several optimistic algorithms for TMP which assume the existence of valid continuous parameters for actions and wait to find a concrete binding for said parameters until all previous steps have been validated. Similarly, we assume that geometric constraints are satisfied until we have obtained evidence that they are invalidated at a plan step. Garrett, Lozano-Pérez, and Kaelbling [39] also offer a notion of optimal TMP, but with regard to symbolic action costs rather than symbolic plan length and motion plan cost.

4.3 Approach

At a high level, our approach is as follows: we plan in a unified state space [110] in which a valid trajectory, annotated with an optional symbolic action at each state, is a valid task-and-motion plan. To find such a trajectory, we use an incremental SAT-based task planning algorithm inspired by Dantam et al. [25] to solve a relaxed symbolic planning problem and generate a candidate symbolic plan. We attempt to sample batches of states [106] in each of the modes (unique settings of symbolic state) traversed by the candidate symbolic plan, using the differentiable geometric predicate representation of Thomason and Knepper [110] to detect when we have found a sample within the

rewire radius [55] of a state that satisfies the precondition for an action in the plan candidate. Only when we are certain that the sampled state will not go to waste (under the assumption that the current plan candidate is feasible) do we perform the relatively computationally costly optimization-based sampling operation to find a precondition satisfying state. If we fail to find valid samples for all the transitions in the plan, we add constraints based on the nature of the the failure to the symbolic planner, request a new plan, and repeat. Otherwise, the AIT* algorithm [106] proceeds unaltered to find a valid motion plan through the sampled transition states to the goal.

We will now proceed to more formally define our problem formulation and approach.

4.3.1 Problem formulation

First, let an *object* $o_i \in \mathcal{O}$ be a physical entity in the environment, associated with a pose $P(o_i) \in SE(3)$ and a 3D geometry. A *surface* is an object that is additionally associated with a 3D bounded region defining an area onto which objects may be stably placed. Let the robot R be defined by its kinematic tree, consisting of links $l_i \in L$ with 3D geometries and joints with joint bounds, as well as its planar base pose $P(R) \in SE(2)$.

Next, let a *predicate* be defined as:

Definition 11: *Predicate* A predicate $p_i \in \mathcal{P}$, $p_i \subseteq 2^{\mathcal{O} \cup L}$ is a relation on a set of objects and/or robot links defining some problem-dependent property. Some predicates, $G_{\mathcal{P}} \subseteq \mathcal{P}$, can be interpreted as defining properties with a reasonable *geometric* interpretation; others, $D_{\mathcal{P}} = \mathcal{P} \backslash G_{\mathcal{P}}$ are best interpreted as defining *discrete* properties.

Finally, let a *proposition* or *symbol* $s_i \in \mathcal{S}$ refer to a specific predicate p_i along with concrete values for its arguments (also referred to as a "ground" predicate). A symbol s_i

is `true` if its arguments are in p_i, and `false` otherwise.

The *state space* for our problem is then:

Definition 12: *State Space*

$$\mathcal{Q} = \mathcal{Q}_R \bigtimes \mathcal{Q}_{\mathcal{O}} \bigtimes_{s_i \in \mathcal{S}} \mathbb{Z}_2$$

where $\mathcal{Q}_R$ is the configuration space of the robot R defined in the usual way [76], $\mathcal{Q}_{\mathcal{O}}$ is the joint pose space of all objects $o_i \in \mathcal{O}$, and $\mathbb{Z}_2$ is a Boolean[1] domain (one "copy" for each symbol $s_i \in \mathcal{S}$, representing its value). We refer to the copies of $\mathbb{Z}_2$ as the "discrete portion" of the state; the remainder is the "continuous portion" of the state. Denote by $D(\boldsymbol{q}_i)$ the set-theoretic projection of $\boldsymbol{q}_i$ to its discrete portion.

As in most symbolic planning and TMP work, we define symbolic models of robot actions in terms of the *preconditions* necessary for the action to successfully execute and the *effects* that executing the action has upon the world state.

Thus, let a *constraint* $\phi_i : \mathcal{Q} \rightarrow \mathbb{Z}_2$ be a deterministic function which evaluates the truth value of a propositional logic formula over symbols $s_j \in \mathcal{S}$ at a state $\boldsymbol{q}_\ell \in \mathcal{Q}$. Note that for symbols corresponding to predicates in $D_{\mathcal{P}}$, the symbol value at a state is determined by the state's value for the copy of $\mathbb{Z}_2$ corresponding to the symbol; we discuss how the values of symbols corresponding to predicates in $G_{\mathcal{P}}$ are evaluated in Section 4.3.2. Further let an *effect* $\psi_i : \bigtimes_{s_i \in \mathcal{S}} \mathbb{Z}_2 \rightarrow \bigtimes_{s_i \in \mathcal{S}} \mathbb{Z}_2$ be a deterministic function which, given the discrete portion of a state, returns a new, potentially altered discrete portion of a state.

An action is then:

[1] Although we restrict ourselves to consideration of Boolean symbols in this work, the proposed planner can be trivially extended to other discrete symbolic domains.

Definition 13: *Action* An action $\alpha_i \in \mathcal{A}$ abstracts a robot controller and comprises a constraint ϕ_i defining its precondition and an effect ψ_i describing its effect on a state $\boldsymbol{q}_j \in \mathcal{Q}$.

We now have the necessary machinery to formally state the planning problem that we solve.

Definition 14: *TMP Problem* Given a set of actions $\mathcal{A}$, an initial state $\boldsymbol{q}_0 \in \mathcal{Q}$, and a goal constraint ϕ_g, the corresponding *integrated task and motion planning problem* is to find a plan: a sequence $[\boldsymbol{q}_0, a_1, \boldsymbol{q}_1, \ldots, a_n, \boldsymbol{q}_n]$, $\boldsymbol{q}_i \in \mathcal{Q}$, $a_i \in \mathcal{A} \cup \{\perp\}$, and $\perp$ signifying the "null action" with precondition $\phi_\perp(\boldsymbol{q}_i) = \texttt{true}$ and effect $\psi_\perp(D(\boldsymbol{q}_i)) = D(\boldsymbol{q}_i)$, such that:

(1) $\forall \boldsymbol{q}_i$, no geometries associated with robot links or objects intersect (i.e., no states are in collision)

(2) $\forall \boldsymbol{q}_i, \boldsymbol{q}_{i+1}$, the motion joining $\boldsymbol{q}_i$ and $\boldsymbol{q}_{i+1}$ is also collision-free

(3) $\forall \boldsymbol{q}_i, \boldsymbol{q}_{i+1}$, for ϕ_{i+1} and ψ_{i+1} the precondition and effect, respectively, of a_{i+1}, $\phi_{i+1}(\boldsymbol{q}_i) = \texttt{true}$ and $\psi_{i+1}(D(\boldsymbol{q}_i)) = D(\boldsymbol{q}_{i+1})$. In other words, the sequence of states and actions comprising the plan respects the transition dynamics imposed by the actions in $\mathcal{A}$.

(4) $\phi_g(\boldsymbol{q}_n) = \texttt{true}$

There are several challenges implied by this problem definition. First, we will need to determine the sequence of actions $[a_1, \ldots, a_n]$ which has the cumulative effect of changing $D(\boldsymbol{q}_0)$ to $D(\boldsymbol{q}_g)$ for some state $\boldsymbol{q}_g$ such that $\phi_g(\boldsymbol{q}_g) = \texttt{true}$, and which is symbolically valid (i.e., all action preconditions are satisfied when their actions are used in the plan). Second, we will need to find continuous portions of states which satisfy the

preconditions of the actions we need to take. Finally, we will need to find collision-free paths between states to reach the precondition satisfying states for each action in the sequence.

We address each of these challenges in the following subsections:

4.3.2 Predicate representation

Recall from Definition 11 that we partition the set of predicates $\mathcal{P}$ into discrete predicates $D_{\mathcal{P}}$ and geometric predicates $G_{\mathcal{P}}$. For some predicates, e.g. a predicate representing a light being on or off, or a predicate representing the distance to an object, this distinction is natural. For others, e.g. a predicate representing an object being located on a specific surface, the choice of whether or not to interpret the relation as a discrete value or as a function on geometric state may be problem-specific. Regardless of how the partition of $\mathcal{P}$ into $D_{\mathcal{P}}$ and $G_{\mathcal{P}}$ is formed, $G_{\mathcal{P}}$ must all have one additional property: they are represented by deterministic, computable, differentiable functions returning the distance from a given state to the nearest state satisfying the predicate. If a geometric predicate $p_i \in G_{\mathcal{P}}$ is $p_i(\boldsymbol{q}_j) = 0$ at some state $\boldsymbol{q}_j \in \mathcal{Q}$, then the predicate holds at $\boldsymbol{q}_j$. Further, by requiring that p_i is differentiable, we can solve the second challenge identified above by using gradient-based optimization [22, 85] to project uniformly sampled states onto the manifold defined by the zero level set of p_i.

Using the "unsatisfaction semantics" of Thomason and Knepper [110] is one way to implement such a geometric predicate function. Unsatisfaction semantics automatically constructs predicate distance functions from Boolean predicates defining combinations of half-spaces on the continuous portion of the state by overloading the standard comparison and Boolean combination (i.e., `and`, `or`) operators. This model of predicate distance

has the additional benefit of being exactly composable for combinations of mutually orthogonal half-spaces, while allowing more general distance functions to be used in isolation. We can alternatively use the semantics of signed distance functions [88] to compose arbitrary distance functions, at the cost of intersection being inexact for some combinations of predicates. Another alternative is the set-based representation of Thomason and Kress-Gazit [111], which can tightly approximate combinations of sets, but for which individual predicates may require more effort to specify.

No matter the form of distance function predicate representation we choose, the composability property means that a propositional logic formula over distance-function predicates is itself a valid distance function. We use this composability to directly sample states which satisfy an action's precondition.

4.3.3 Symbolic planning

To solve the first challenge identified above (finding a sequence of actions which accomplishes the symbolic component of the goal), we develop an incremental SAT-based symbolic planning algorithm adapted from Dantam et al. [25].

The core idea of SAT-based symbolic planning is to (1) encode the symbolic problem state and each action as a Boolean variable, (2) add constraints ensuring that variables remain consistent between steps (i.e., the "frame axioms") and that actions imply their preconditions and effects, and (3) iteratively add new "steps" at which exactly one action variable must be true, until the resulting formula is satisfiable [57, 59].

Our implementation follows Dantam et al. [25] in many places; we will highlight the significant differences.

State encoding

As in Dantam et al. [25], we create one Boolean variable for each symbol at each step of the plan. However, because of our partition of the predicates into discrete and geometric, we *only* create variables for the discrete symbols (i.e., those corresponding to predicates in $D_{\mathcal{P}}$). We additionally do not include symbols for the geometric state of the problem at a step, differing from Dantam et al. [25]. This effectively creates a relaxed symbolic planning problem in which constraints induced by geometric predicates are optimistically ignored. Let s_i^j refer to the Boolean variable for the value of symbol s_i at plan step j.

Action encoding

Our naive action encoding is, again, similar to Dantam et al. [25]: each action α_i gets a Boolean variable α_i^j representing whether we choose α_i at step j by its value. We add clauses such that $\alpha_i^j \implies \phi_i^{j-1} \wedge \psi_i^j$, where ϕ_i^{j-1} is the discrete part of the precondition of α_i encoded using the variables for the relevant discrete symbols at step $j-1$, and ψ_i^j is the effect of α_i applied at step j. These clauses enforce that an action can only be chosen at a step if the discrete part of its precondition is satisfied after the preceding step, and that choosing an action changes the state accordingly for the step at which it is "run".

We extend this action encoding to support more efficient plan search in Section 4.3.5.

Frame axioms and action mutexes

Next, we need to add the frame axioms and action mutex constraints. The frame axioms specify that:

$$\forall s_i^j : s_i^{j-1} = s_i^j \vee \left(\bigvee_{\alpha_k \in \mathcal{A}_{s_i}} \alpha_k^j \right)$$

where $\mathcal{A}_{s_i} \subseteq \mathcal{A}$ is the set of actions which modify s_i in their effects. In other words, a symbol's value at step j must be unchanged from its value at the preceding step unless an action that modifies it was chosen at that step. This ensures that variable values remain consistent between steps.

Finally, we add action mutex constraints to ensure that exactly one action is chosen at each step. These constraints take the form:

$$\bigwedge_{\alpha_i \in \mathcal{A}} \left(\alpha_i^j \implies \bigwedge_{\alpha_k \in \mathcal{A}; k \neq i} \neg \alpha_k^j \right)$$

Problem encoding

The problem is thus specified using the above encodings: We first add, at step zero, the values of all discrete symbols in the initial state using variables as defined in Section 4.3.3. We then add, for each of $n \geqslant 0$ steps, action variables and clauses for each action per Section 4.3.3, as well as the frame axioms and action mutex clause from Section 4.3.3. Finally, we add variables corresponding to the desired values for the discrete part of the goal at step n.

The symbolic planning loop

With the problem encoded as above, the symbolic planning loop proceeds as shown in Algorithm 3, which is essentially the same as the first inner loop of Algorithm 1 in Dantam et al. [25].

Algorithm 3: Iterative symbolic planning

Input: Set of clauses ρ, step counter n, set of actions $\mathcal{A}$, initial state $\boldsymbol{q}_0$, goal ϕ_g
Output: Action sequence $\mathcal{A}'$, modified step counter n

if $\rho = \varnothing$ **then** `/* Initialize the planning problem */`
  $\rho \leftarrow D(\boldsymbol{q}_0)$
  `push(`ρ, $D(\phi_g)^0$`)`
$\sigma \leftarrow \varnothing$
while $\sigma = \varnothing$ **do** `/* Iterate until satisfiable */`
  $n \leftarrow n + 1$
  `pop(`ρ`)` `/* Pop the goal constraint */`
  $\rho \leftarrow \rho \wedge$ Action clauses from $\mathcal{A}$ at step n
  $\rho \leftarrow \rho \wedge$ Action mutex constraint at step n
  $\rho \leftarrow \rho \wedge$ Frame axioms at step n
  `push(`ρ`)` `/* Push the current set of constraints */`
  `push(`ρ, $D(\phi_g)^n$`)` `/* Push the goal constraint */`
  $\sigma \leftarrow$`solve_SAT(`ρ`)`
return `extract_plan(`σ`),` n

In Algorithm 3, `extract_plan()` is a subroutine which returns the list of actions corresponding to the action variables which are true at each step. `push()` and `pop()` manipulate the constraint stack of the SMT solver we use to solve this SAT problem via `solve_SAT()`. Using the constraint stack in this manner accelerates solving for symbolic plans when incrementally adding steps and (as discussed in later sections) adding constraints to generate alternative plans by avoiding redundant work for steps which do not need to change.

4.3.4 Integration with AIT*

With the basic form of our symbolic planner defined, we can proceed to the motion component of our planner, which drives overall TMP solving. As mentioned, we base our planner around a small modification to the AIT* asymptotically optimal batch-sampling-based motion planner [106]. At a high level, we modify AIT*'s batch sampling function to (1) generate a batch of samples in each mode we know how to reach, and (2) sample states which satisfy action preconditions if and only if a uniform sample is within the rewire radius [55] of the action precondition region.

A mode—which is defined by a unique setting of the discrete portion of the state and a unique set of poses for all movable objects in the environment—is *reachable* if it is the initial mode or if we have successfully sampled at least one state which satisfies the precondition of an action that, taken from the mode of the precondition-satisfying sample, results in the mode we wish to reach. The intuition for the second modification we make to AIT* is that, as in many TMP problems, when a motion planning problem is hard (i.e., highly geometrically constrained), we do not wish to waste effort on the relatively expensive precondition-satisfying state sampling procedure until we have some confidence that the results will go to use. Testing the distance from a state to the nearest precondition-satisfying state costs about one to two orders of magnitude less computation than sampling a precondition-satisfying state, on average (assuming on the order of 10-100 iterations of a numerical optimization method for the projection-based approach we take to satisfying precondition-satisfying states). This is particularly helpful as sampling continues after finding an initial solution, since it helps us avoid sampling precondition-satisfying states unless they could possibly improve our solution quality. Further, although the manifolds of precondition-satisfying states for many actions may be measure zero in the ambient configuration space (and thus we will naively sample

them with probability zero), by projecting a uniform sample onto the precondition-satisfying manifold whenever we are within the rewire radius, we effectively *inflate* the precondition-satisfying manifold.

We show pseudocode for the modified batch sampling function in Algorithm 4.

Algorithm 4: Multimodal batch sampling

Input: Batch size b, mode queue ω, rewire radius μ, goal constraint ϕ_g
Output: Batch of samples B

```
B ← {}
while |ω| > 0 do                          /* Iterate all reachable modes */
    mode ← pop(ω)
    while |B| < b do                      /* b samples per mode */
        s ← sample_valid(mode)
        B ← B ∪ {s}
        if φ_g(s) then
            return                        /* Return early if goal reached */
        for α_i ∈ mode.actions do
            if should_attempt(α_i) and distance(φ_i, s) < μ then
                s' ← sample_constraint(φ_i, s)
                if is_valid(s') then
                    B ← B ∪ {s'}
                    update_reachable_modes(ω, α_i)
if no_remaining_actions then
    increase_sampling_threshold()
if ¬reached_mode(B, D(φ_g)) or no_remaining_actions() then
    update_task_plan()
return B
```

In Algorithm 4, `sample_valid()` is a standard state sampler for states in the valid configuration space of a given mode. `should_attempt()` checks if we have failed or succeeded at a given action more than a heuristic threshold of times, indicating that we should give up on sampling states from which to take the action. "Attempting" an action means trying to sample a valid precondition-satisfying state for its precondition constraint. `increase_sampling_threshold()` increments the threshold used by `should_attempt`. `no_remaining_actions()` tests if we have actions

in any reachable mode for which `should_attempt()` returns true. `distance()` computes the distance from a sampled state to the nearest constraint-satisfying state, and `sample_constraint()` uses gradient-based optimization to project the given state onto the manifold of constraint-satisfying states. `is_valid()` checks if a state is collision-free, `update_reachable_modes()` adds the newly reached mode to the mode queue, `reached_mode()` checks if the set of samples contains states in the given mode, and `update_task_plan()` invokes the task planner as described in Section 4.3.5. The mode queue is ordered such that modes reached by a recently successful transition are prioritized over modes which were most recently reached by older successful transitions. This ordering has the property of creating behavior akin to the "enforced hill climbing" of the Fast-Forward task planner [51]: when we succeed at a transition, we continue the search in the resulting mode, effectively following the corresponding task plan candidate as far as we can. Although this heuristic may cause us to over-explore unsuccessful candidate plans with "easy" prefixes, the incrementally-raised cap on action transition samples means that newer candidates will be more likely to have transition sample budget remaining, and therefore we will be more likely to attempt and succeed at the transitions corresponding to newer candidate plans. Thus, the mode queue's ordering keeps the planner focused on promising plans. This focus pays off due to the early return condition if we reach a goal-satisfying state.

4.3.5 Considerations for using AIT* in multimodal spaces

AIT* was not originally designed for use in multimodal spaces; as such, we need to make a handful of smaller modifications to its search algorithm and data structures. Most importantly, transitions between modes in our space are *directed*: the existence of an action which moves from mode m_i to mode m_j does not imply the existence of an action

which moves from mode m_j to mode m_i. This directedness is important for the reverse search of AIT*, which estimates a heuristic value for states in part defined by the cost of the straight-line (in configuration space) edge joining them [106]. If two states exist in different modes with a transitioning action in only one direction, then the edge between them will have finite cost in the direction of the transition, but infinite cost in the reverse direction. Because the reverse search moves from goal states toward initial states, and because transitions in the forward symbolic plan move from initial states toward goal states, all edges between modes falsely appear to have infinite cost. Fortunately, we trivially work around this issue by reversing the edge direction when computing the cost in the reverse search (as this has no effect on the cost of edges within a single mode).

Additionally, because we are manipulating multiple objects, we initially have no way of knowing the possible poses of objects at the goal. This motivates our use of a reachable mode queue, which, for the initial task plan, forces batch sampling to proceed from the initial mode toward the goal mode. Thus, as we can find a new valid object pose set at every mode transition, the first (successful) forward pass of sampling gets us at least one set of possible object poses in the goal mode, enabling us to (if needed) directly sample goal states.

Finally, computing exact distance in our multimodal space is equivalent to solving the symbolic planning problem, and is thus PSPACE-complete [56, 116][2]. We instead use a conservative overapproximation of distance wherein states in different modes are infinitely far apart if we do not know a path between their modes. This approximation, combined with the aforementioned directedness of mode transitions, means that distance is at best a quasimetric, but this works well in practice.

[2]Intuition: to know the distance between two states, you must know the shortest sequence of mode transitions between the states, which requires finding an optimal symbolic plan

Generating alternative plan candidates

Core to any TMP algorithm is its method for finding alternative symbolic plan candidates when a previous candidate is found to be infeasible. As shown in Algorithm 4, we detect an infeasible plan when we fail to sample all the way to the goal mode during a batch; this implies that a transition (corresponding to an action in the symbolic plan) failed. When this occurs for a plan with n steps, we add the following constraint to naively force the symbolic planner to return a new plan:

$$\neg \bigwedge_{1 \leqslant j \leqslant n} \left(\bigwedge_{\alpha_i \in \mathcal{A}} \alpha_i^j = \texttt{value}(\alpha_i^j) \right)$$

where $\texttt{value}()$ returns the value assigned to a variable in the current solution. This constraint forces the next plan to differ in at least one action from the current plan, and matches the approach taken by prior constraint-based symbolic planners [25, 57, 58].

By iteratively applying these constraints and replanning as needed, we gradually enumerate all possible symbolic plans for the problem.

Extensions to reduce the action search space intelligently

Although the constraints proposed in the preceding section will permit us to enumerate alternative plans, this search process can be inefficient in problems with a large number of candidates to process. We want to more aggressively filter the space of candidate plans. To do so, we introduce the notion of *conditionally included geometric constraints*. Recall that we solve a relaxed symbolic planning problem that optimistically ignores geometric constraints in preconditions. Doing so is beneficial for the difficulty of the symbolic planning problem, but throws away information about plan feasibility. The other motivation for dropping the geometric constraints is that—as a tradeoff for their

relatively easy sampling—they may be expensive or intractable to compute during task planning. Consider the example of determining if the path between an object and a manipulator is occluded by any other objects: to do so, we would need to first know the object and robot pose in the symbolic planning problem, and then compute a swept-volume collision problem, for every updated plan. While an approach of this sort is technically possible [27], it is in practice inefficient and rapidly intractable for continuous sets of poses. Instead, we use evidence from the state sampling process to decide whether or not continuous constraints are relevant at a given step.

Revisiting the encoding of Section 4.3.3, let us now add to the precondition clause of each action clause the following structure: for each geometric symbol s_i in the precondition of an action α_j, substitute $(b_i^k \vee s_i^k)$, where b_i^k is a *blocker variable*. A blocker variable is a Boolean variable which is initialized to `true` and prevents the value of its associated geometric symbol from mattering for a solution. Thus, by default, we are still solving the same relaxed symbolic planning problem. However, we additionally modify each action's effect clause to add clauses of the form $s_i^{k-1} \implies b_i^k$ for every geometric symbol s_i which we know α_i to modify. In other words, if the geometric symbol is unblocked, this action puts it back into an "unknown" state by blocking it again.

The final piece of this mechanism is to add constraints to unblock geometric symbols. In general, detecting that a geometric condition holds or does not hold requires a condition-specific decision procedure, as in Srivastava et al. [105]. However, we can detect some common conditions based off of collision checking results. Specifically, we can use the output of state validity checking and motion validation to heuristically detect when one object occludes another, or if the locations for placing an object are blocked by another, etc. When we have detected a specific geometric condition, we unblock its symbol for

the specific plan prefix that caused it. Given a geometric symbol s_i found to be true after a sequence of actions $\alpha^1_{i_1}, \ldots, \alpha^n_{i_n}$, we add the constraint:

$$\left(\alpha^1_{i_1} \wedge \ldots \wedge \alpha^n_{i_n}\right) \implies \left(\neg b^n_i \wedge s^n_i\right)$$

The intuition for using blocking variables rather than simply setting s_i to `true` or `false` is that we do not know which value is "optimistic" for each geometric condition in each precondition formula. By instead blocking and conditionally activating the variables, we can rule out classes of actions based on geometric constraints without needing this information or needing to track a different optimistic value per action.

4.3.6 Implementation

We have implemented a proof-of-concept planner using this approach in C++. We build off of the Open Motion Planning Library [108] for their implementation of AIT* and associated sampling-based planning machinery, and use Bullet [23] for collision checking. Our predicate implementation is an improved version of the system in Thomason and Knepper [110] using Autodiff [79] for automatic differentiation and an improved bespoke dual number automatic differentiation implementation in LuaJIT for predicate implementations. Our symbolic planner is based on the Z3 SMT solver [26].

The architecture of the resulting system is very similar to that shown in Fig. 2.1. We take in a JSON-based description of the initial scene, URDF specifying the robot morphology and kinematics, and mesh files describing object and robot geometries. The symbolic problem and its domain are specified as PDDL augmented with annotations for separating discrete and geometric predicates. The planner grounds the PDDL domain to the given problem and constructs an appropriate composite state space for the resulting discrete predicates, movable objects, and robot kinematics. Predicate semantics are, as

in Section 2.3, loaded as Lua functions to be evaluated with LuaJIT. The planner bridges its internal state with LuaJIT via the latter's foreign function interface for efficient data structure sharing during precondition region distance testing and precondition-satisfying state sampling.

4.4 Analysis

We will now sketch proofs of probabilistic completeness and almost-certain asymptotic optimality for this approach. Because we use projection from uniform random states to sample in precondition-satisfying manifolds, we must assume that the precondition regions are all convex for these proofs. This restriction could potentially be lifted by using a different sampling mechanism, or perhaps by sampling from the predicate formulation in Thomason and Kress-Gazit [111].

Theorem 1: *Probabilistic Completeness* For any TMP problem, our approach will find a solution per Definition 14 if one exists with probability approaching one as the number of samples taken approaches infinity.

Proof. First, our base symbolic planner is known to be complete [58]. We always eventually enumerate all possible symbolic plans by adding the constraints discussed in Section 4.3.5. We invoke the symbolic planner infinitely often as the number of samples goes to infinity, as we request a new task plan whenever either we fail to reach the goal mode or we have attempted each action more than the `should_attempt()` threshold number of times. The latter event will happen infinitely often: even though we exit the sampling loop early when we reach a goal-satisfying state, all transitions that reach the goal will eventually run out of sampling budget until the next cap increase, and the goal mode is otherwise last in the mode queue, meaning that we will attempt all

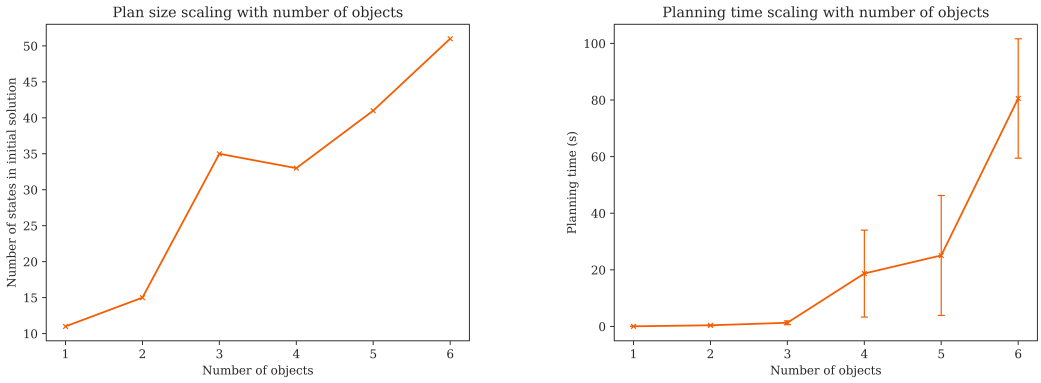
other known transitions in all other known modes before exiting. Thus, we will attempt every possible symbolic plan candidate. AIT* is itself probabilistically complete [106], and we will eventually attempt every action infinitely often. As we have assumed that the precondition-satisfying manifold for each action is convex, and we are projecting uniformly random samples onto said manifolds, we will eventually sample every point in the manifold. Thus, we will find a valid path through our planning space if one exists, and therefore are probabilistically complete. □

The proof sketch for almost-certain asymptotic optimality is similar: we will eventually enumerate and evaluate all candidate task plans for a given problem, AIT* is asymptotically optimal within each mode, and because the AIT* reverse search heuristic is computed globally, AIT* will choose the optimal sequence of mode transitions. Finally, by the previously-made assumption of precondition region convexity, we will eventually sample every precondition-satisfying state for every viable action. Because we sample said states when they are within the rewire radius of an "ordinary" sample, we will add new action transitions whenever they would improve the local solution cost. Under the assumption that the rewire radius never reaches zero, this has the effect of "inflating" even zero-measure precondition regions to have non-zero sampling probability in the ambient state space. Thus, as the number of uniformly distributed "ordinary" samples approaches infinity, the probability that we will not sample states corresponding to the optimal task plan transition sequence approaches zero.

4.5 Evaluation

We evaluate on the domains used in Thomason and Knepper [110] to make the comparison of the two methods as fair as possible. Specifically, in Fig. 4.1b we show the total planning

time for instances of the clutter problem used in Section 2.5. Figure 4.1a shows the average plan length for each problem instance size. The data shown are averaged across ten trials for each size, and plotted with error bars showing one standard deviation from the mean.



(a) *(b)*

Figure 4.1: Performance results on the "Clutter" problem

In comparison to the results shown in Fig. 2.3b (specifically, the time data for the clutter problem without red blocks), we substantially outperform Thomason and Knepper [110] on overall planning time and on the variance of planning time (note that whereas the y-axis scale of Fig. 2.3b is logarithmic, the y-axis scale of Fig. 4.1b is linear). The comparison on plan length versus the results of Fig. 2.3a is less clear, but we regardless show a linear trend in the growth of the size of the plan produced by our proposed method, as desired.

Figure 4.2 shows the output from an instrumented run of our proposed planner, highlighting where the planner spends time.

Surprisingly, most of the planner's time is spent generating new symbolic plan candidates. This result suggests that as the symbolic model size grows (i.e., more objects means more ground actions and predicates), the time for symbolic planning increases

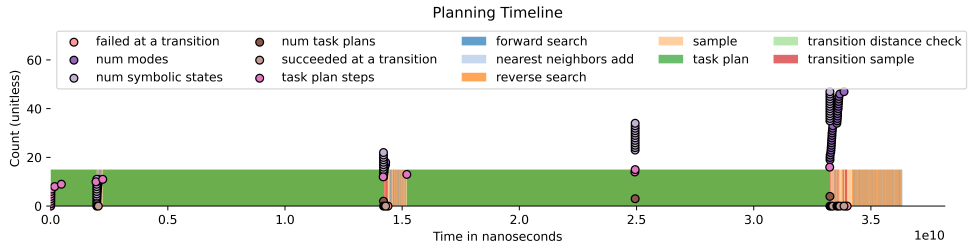


Figure 4.2: The timeline of one run of a size five instance of the "Clutter" problem.

faster than the time for motion planning, leading it to dominate the overall planning time. This result runs counter to the observations of Dantam et al. [25], and motivates further work on developing a more efficient problem encoding for the symbolic planner. Also noteworthy in Fig. 4.2 is the relative amount of time spent sampling transitions versus checking the distance to a transition. The latter category is effectively invisible on the timeline, whereas the former can be seen. This suggests that, as expected, the time required to sample a precondition-satisfying state is non-trivial compared to the amount of time required to test the distance from a state to the nearest precondition-satisfying state, and thus that our proposed method of deferring transition sampling until the motion planning graph is close enough to use the transition sample is successful at avoiding wasted computational effort.

4.6 Conclusions

This chapter contributes a novel approach to TMP which builds upon earlier work in efficient constraint-based symbolic planning for TMP [25], the use of a distance-function-based representation for geometric predicates for efficient action precondition sampling [110], and batch-sampling-based optimal motion planning [106]. Our proposed approach is able to defer computationally expensive sampling effort until it is needed and rule out classes of plans based on conditionally included geometric information.

Batch-sampling-based motion planning, or lazy motion planning more generally, is well-suited to TMP problems because it provides information about action feasibility (via state validity checking) early, before paying the full cost of validating a plan's edges. Further, by incrementally improving a random geometric graph in a unified hybrid state space, we can reuse motion planning effort across multiple symbolic plan candidates.

In future work, it would be worthwhile to investigate using a stronger notion of informed sampling to bias growth in the RGG toward task-relevant regions. This could accelerate the discovery of an initial solution.

CHAPTER 5

CONCLUSIONS AND DISCUSSION

This dissertation has contributed a novel perspective on integrated task and motion planning, or TMP, based around constructing a holistic planning space including both symbolic and geometric problem state, as well as a representation for predicates with continuous geometric interpretations via differentiable distance functions. We have used this perspective to build two task and motion planners (Chapters 2 and 4) which offer competitive performance without requiring as much manual engineering effort as alternatives, as well as to frame and provide an initial approach to the automatic symbolic-geometric abstraction repair problem (Chapter 3), a means of lightening the burden of domain expert engineering necessary to create symbolic abstractions for efficient TMP.

Ideas related to elements of this perspective have been proposed in the literature; work such as Plaku and Hager [90] and Plaku, Kavraki, and Vardi [91, 92] uses a task planner to guide a motion planner to task-relevant subsets of the configuration space, and prior work in manipulation planning [16] and multimodal motion planning [48] has used similar hybrid state spaces to ours. Contemporaneous work [65] has adopted similar ideas around online reweighting of mode transitions to guide a motion planner in a multimodal space. Our particular formulation of a unified state space—separating out symbols by their interpretation (i.e., the discrete part of our hybrid state only includes purely discrete symbols, and symbolic relations with continuous interpretations are implicitly represented via transition sampling)—is unique, and offers benefits for TMP by simplifying the symbolic planning problem, providing samplers for precondition-satisfying states "for free", and resulting in a structure amenable to a high degree of factoring—and thus easier motion planning effort reuse—by values of the discrete state. Moreover, selectively framing predicate relations with continuous interpretations as differentiable functions

computing the distance to the nearest relation-satisfying state is a powerful, novel fusion of ideas from constrained planning and TMP. Such functions can be defined or discovered in many ways (e.g., via composable primitives, as in Chapters 2 and 4, by manual engineering, as various classes of learned models—which may themselves serve as composable primitives), and this paradigm of predicate representation enables richer integration between motion planning algorithms and symbolic planners than most other approaches to grounding symbolic relations in physical states (e.g., because the distance function can serve as both a source of guidance and a state-sampling mechanism, while avoiding the computational cost of taking a full constraint-satisfying sample). Moreover, this predicate representation is complementary to the use of more specialized samplers for action precondition-satisfying states. The distance function representation is useful as a guiding signal for a motion planner in hybrid state space, and any given specialized sampler can then be used to sample an actual precondition-satisfying state only when the motion planner is "ready" for it, i.e., when the state has a high probability of belonging in a successful solution plan.

The planners we contribute offer competitive performance with less information (i.e., constraint network annotations, specialized samplers) than alternatives. Our focus on viewing TMP as a problem of motion planning between task-determined subgoals (which is not new in itself) offers benefits when geometric constraints are the primary source of problem complexity: action parameters will implicitly be selected based on motion reachability, rather than needing to reattempt actions to discover a reachable set of parameters. Indeed, a major component of this perspective is the idea that continuous action parameters are implicitly represented by states, rather than the other way around (i.e., explicitly including continuous action parameters in the planner state model). This component is valid for the kinds of actions we consider here, but would require extension for actions with state-independent parameters, e.g., durations, torques, etc. rather than

poses and other parameters that can be computed directly from state values.

5.1 Possible extensions

This dissertation leaves some low-hanging fruit unexplored. In this section, we attempt to describe some of the more promising potential extensions to the ideas of this dissertation.

5.1.1 Reusing state samples across modes

The planners in Chapters 2 and 4 rely on growing motion planning graphs in each mode (unique setting of the discrete state) and connecting these graphs via transitions corresponding to executing symbolic actions. They consider sampled states as separate for each mode, given that the valid configuration space may be different in different modes. This is coarsely correct, but it is not always true that different modes have different valid configuration spaces: not all changes in discrete state correspond to a difference in the kinematic graph of the environment or the constraints of the system. Thus, if we can inexpensively identify equivalent kinematic graphs (for some problem-dependent notion of equivalence), we can reuse samples—and their corresponding planning graphs—between different modes. This would avoid redundant sampling and collision checking effort and increase planner performance.

5.1.2 Parallel TMP

The planning structures that our planners create are only coupled at mode transition states. Sampling, including of action parameters and connecting states, is otherwise completely

independent between modes. This structure suggests that we could perform sampling in parallel across multiple modes simultaneously. Parallel sampling in this manner not only linearly accelerates TMP solving by letting us evaluate several task plan candidates at once, but also dovetails nicely with online planning, in that we can explore the later stages of multiple plan candidates in parallel while executing a common plan prefix.

5.1.3 Implicitly defined constraint networks

Finally, as we will discuss below, one of the main advantages that planners like Garrett, Lozano-Pérez, and Kaelbling [39] offer over the planners of Chapters 2 and 4 is that, once their higher upfront specification cost has been paid, their use of an explicit constraint network can lead to faster planning by filtering out invalid plan and action parameter candidates based on the choices made at earlier stages. These constraint networks exist for all problems that admit a factorable structure—for the perspective and planners considered in this dissertation, the constraint networks are defined implicitly through the precondition formulae of the actions available to the planner. If we could automatically discover the structure and necessary invariants of these implicit constraint networks (e.g., via symbolic execution to gather information about bounds on/invariants of the inputs and outputs of the continuous predicate implementations), we could construct a planner that exploits the best of both worlds: rich constraint information with low specification effort overhead.

5.2 The state of TMP, and looking forward

Modern TMP is starting to become practically useful for a narrow subset of the problems for which TMP shows potential: fully-observable quasistatic environments with rich, correct, deterministic symbolic action models. There are several reasonable choices for planners for these domains [24, 39], arguably including the contribution of Chapter 4. While planner efficiency likely can still be significantly improved, TMP performs reasonably well under these strong assumptions.

5.2.1 Where should TMP go from here?

With that said, TMP has a long ways to go before becoming practically useful outside of the lab at the level of either of its core components (pure motion or task planning). Mansouri, Pecora, and Schüller [82] take a forward-looking view of the area, noting in their selected questions for guiding the use of TMP in real-world problems that deciding how to enable online planning and planning under uncertainty for TMP cannot be done in isolation from questions of problem abstraction and joint reasoning between discrete and continuous representations. Here, we wish to highlight some prominent desirable properties which TMP currently lacks, as well as briefly discuss recent work starting to address these shortcomings and some potential routes forward.

TMP under uncertainty

As mentioned repeatedly in this dissertation, most TMP work to date has operated in fully-observable, quasistatic environments with deterministic actions (with a handful of notable exceptions [53, 54, 107]). This is untenable for most of the unstructured,

real-world environments in which we want robots to operate. In these settings, there may be uncertainty in observations (i.e., in partially-observable environments) or in execution (i.e., with nondeterministic action controllers). Efficiently and effectively handling these sources of uncertainty in TMP is crucial for practically useful TMP. The problems caused by the two sources of uncertainty are closely related; both forms of uncertainty may necessitate finding a valid branching plan to handle contingencies or modifying a plan online in response to an unexpected event. Some recent work has started to think about these sources of uncertainty—primarily execution uncertainty, which is typically handled by policy synthesis [102, 120]. Work that accounts for observation uncertainty tends to plan with "belief states" representing a distribution over possible world states, determinize action dynamics, and iteratively replan as plan execution results fail to match expectations [41, 53].

This replanning-oriented style of approach can be an effective route toward accounting for, in particular, execution uncertainty. Whereas policy-based handling of execution uncertainty is only effective with a large amount of information about the possible distribution of action effects and state visitation, as well as long planning time to generate a full policy, replanning can offer better overall efficiency (if replanning is cheap and/or infrequent) and flexibility (e.g., to account for unmodeled action effects). Naive replanning is often inefficient; one potentially interesting path toward solving this problem is to replan from an unexpected deviation back to the original plan (if possible), by e.g., warm-starting trajectory optimization with the intended plan, or continuing to grow an existing motion plan graph generated during the initial planning stage. In all considerations of replanning for TMP, there is also a fundamental problem of deciding what aspects of the plan must be changed (i.e., only low-level motions or also high-level plan structure). Deciding what to reuse and what to replace more efficiently than re-solving the planning problem from scratch remains an important, open problem for TMP replanning. The ideal

approach to TMP under uncertainty may involve a fusion of policy- and replanning-based methods, wherein partial policies are computed for subproblems which are well-modeled and hard to predict, and replanning fills in the gaps between these subproblems, where modeling information is more sparse or execution failure less probable.

Reducing the specification burden of TMP

The biggest obstacle to practical adoption of even fully-observable quasistatic TMP is its need for expert-created symbolic action models and related components (e.g., action parameter samplers, constraint specifications), which goes beyond that required for ordinary task planning (already a high burden for many applications). These models must be correct for TMP to work, and creating a correct model often requires significant effort involving both domain expertise and expertise with symbolic planning methods. Further, choices of symbolic model abstraction can have an impact on TMP performance by making the symbolic planning problem harder or easier, or by providing abstractions that correspond more or less closely to the physical problem. Work like that in Chapter 3 [111] attempts to mitigate this burden by allowing partially incorrect or incomplete symbolic models to become more useful over time, but this first step is insufficient. Other recent work attempts to learn action parameter samplers [118] and symbolic action models [68, 119, 121] directly, but this work still makes strong assumptions about access to some level of symbolic specification at the start of the learning process, requires a potentially large number of expensive observations to build a correct model, and is limited in the complexity of the action constraints that it can learn. Further work in these directions will go a long way toward making TMP practically useful; any contributions that lower the specification effort required for TMP at no (or minimal) cost to performance help move toward this goal.

"Have you tried a neural net for that?" (TMP and learning)

Finally, although TMP has thus far been a primarily "classical" (i.e., non-learning-based) approach to robot autonomy, there is substantial untapped potential for advancing robot autonomy by combining TMP with robot learning. Each of TMP and robot learning for control have a property that the other lacks and needs: TMP is generalizable to new domains and problems without retraining or retuning, and can handle long-horizon reasoning, whereas learning may not require as much manual effort to get up and running, and may be better at handling difficult-to-model actions. As Garrett et al. [36] note, there are several fruitful ways in which learning and TMP can be (and already are) combined. Symbolic model learning, as discussed above, is perhaps the most desirable opportunity for learning to benefit TMP. One underexplored direction in this space is that of using a target symbolic model to guide action controller learning, rather than the other way around: with a notion of continuous predicate interpretation as proposed in this dissertation, one could generate a training signal for an action controller to match a given action model specification. Constraining the action search problem in learning in this way could accelerate the learning process, with the resulting learned controller ready to be used for planning. Some work in the formal methods community has started to look at discovering "useful" missing controller specifications [89], which could serve as the input to such a skill learning method. Separately, recent work in using learned heuristics to guide TMP [61, 62] has also shown promise, and may be a way forward for increasing TMP search efficiency. Finally, it would be interesting to attempt to learn TMP-amenable perceptual representations by embedding a metric of TMP planner effectiveness in the objective for a model learning to abstract high-dimensional raw observations to a low-dimensional abstract state space. This sort of state space co-design presents an interesting chicken-and-egg problem: the best state space for a TMP planner depends in part on the discrete problem abstraction (i.e., the symbolic model) it is given, but the best symbolic

model for a problem also depends in part on the properties of the planner's state space.

5.3 Notes on TMP system design and implementation

Each of Chapters 2 to 4 has entailed designing and implementing a complex system for TMP or abstraction repair. As such, this section includes brief notes on the design choices and other system-building considerations that went into this process and may be useful for others working on similar tools.

5.3.1 Automatic differentiation

The work in Chapters 2 to 4 makes heavy use of *automatic differentiation* techniques to compute gradients for optimization subproblems. Automatic differentiation (AD) is a family of methods for computing derivatives and related quantities of a broad family of functions with little to no programmer effort beyond specifying the function to be differentiated. AD has seen a large upswing in adoption among machine learning communities in recent years, as both AD techniques and practical implementations thereof have become abundant, powerful, and efficient. Traditional areas of robotics (e.g., motion planning and inverse kinematics) use explicitly computed, often manually-defined Jacobian matrices as intermediate values in solving for related gradients and other derivatives. Neither analytical Jacobians nor automatic differentiation is, in our opinion, a clear universally superior solution to this class of subproblem. The two techniques compete on several tradeoffs: convenience, efficiency, and extensibility. The clearest advantage of automatic differentiation is its convenience: in commonly used programming languages (specifically Python and C++), using an automatic differentiation

library such as Autodiff [79] or Jax [13] to compute derivatives, gradients, and higher-order differential values requires nothing beyond calling a generic function with an appropriate datatype (for forward-mode automatic differentiation, which is preferable for the applications considered in this dissertation). The resulting derivatives will be exact and fast to compute. Although modern kinematics libraries, such as Pinnochio [17], can provide a similar level of convenience for dynamically computing various Jacobians of a robot's kinematics function, AD libraries win out the moment one wants to do something more advanced. For example, when the gradient is needed for optimizing a potentially complex function of the robot kinematics—such as for continuous predicate definitions of the sort discussed in this dissertation—computing the final gradient with AD is no different than computing the basic manipulator pose Jacobian. In contrast, using a traditional Jacobian library will still require manually deriving the remainder of the gradient in terms of the Jacobian. While this is not always a great challenge, the difference in convenience and extensibility can add up to a simpler-to-design, more flexible system. Moreover, if higher-order derivatives are called for (such as with predicates on dynamic properties of robot motion), AD tools are usually well-equipped with no more effort than to compute a first-order derivative; Jacobian libraries often do not support automatic higher-order derivatives. The other side of these considerations is efficiency. Modern automatic differentiation techniques are fast, and should often be faster than dynamically computed Jacobians for many optimization objectives: if the objective is sparse in the full Jacobian (i.e., the objective function only depends on a small number of the Jacobian's elements), then AD techniques have the potential to skip computing unnecessary terms. Moreover, dynamically computed exact Jacobians use much the same techniques as forward-mode automatic differentiation, so the computational costs of the two are comparable. However, modern Jacobian libraries [17] support ahead-of-time generation of closed-form expressions for specific Jacobians, which—primarily by virtue

of benefitting from compiler optimizations—can be significantly more efficient than dynamic computations[1]. As such, TMP systems which make use of kinematics gradients should consider whether the convenience and extensibility of using AD for differentiation is worth the performance hit in comparison to a statically-generated Jacobian expression.

5.3.2 State design and implementation

Although perhaps particular to the composite state spaces used in this dissertation, designing and implementing the planning state space for each system has been pivotal for the efficiency and ease of implementation of the rest of the system. Part of the need for this care is that motion planning libraries [108] are not designed with TMP in mind as a first-class citizen; provided features for motion validation, collision checking, and nearest-neighbors search implicitly assume that they will operate in a single fully continuous state space. These assumptions are not a major impediment to TMP, but they do require that a larger-than-normal amount of functionality be redefined (compared to implementing a new proposed motion planning algorithm, etc.). One TMP-specific decision that proved important was the choice of representation for movable object poses. In both Chapters 2 and 4, we use scene graphs—generalized kinematic trees—to represent the environment and robot. As TMP problems frequently involve manipulating movable objects, and as the collective poses of these objects correspond to different instantiations of continuous action parameters, it is important that the representation used for movable object poses is efficient and easy to use both for objects which are currently static and objects which are currently being manipulated. Scene graphs are a natural choice for this representation:

[1] This leads to an interesting line of thought fusing optimizing just-in-time (JIT) compilation with automatic differentiation for robotics. This technique is already employed by some libraries [13], but is not, to our knowledge, widely employed or specialized for robotics.

they admit nodes representing robot links, static obstacles, and movable objects in the same abstraction, and represent relationships such as objects holding other objects (with the associated notion of collective motion) similarly naturally. In Chapter 2, this representation was used crudely. Object poses were separately encoded in the standard $SE(3)$ representation and included in individual states; while scene graphs were used to update movable object poses during forward kinematics, this necessitated repeatedly determining which objects were currently being manipulated, as well as tracking both a large amount of pose information in each state and maintaining an association between each state and its corresponding scene graph. The work in Chapter 4 improved upon this state of affairs. As a single scene graph corresponds to a single environment state (i.e., a particular free configuration space), we can instead include only a unique identifier of the scene graph associated with a state in said state—in other words, tracking only which scene graph should be used to compute object poses. This reduces the amount of bookkeeping and the size of each state, as well as the computational effort of copying object poses back and forth between two semi-redundant representations.

Separate from the representation of movable object poses is the representation of the discrete bits of the state—a necessary part of TMP. Here, bitsets proved useful for efficiently testing states against symbolic action preconditions and propagating symbolic effects of actions. The more salient design choice was to represent discrete state outside of the system of state abstractions offered by existing motion planning libaries. This choice may seem to go against the philosophy of crafting a holistic state space for TMP, but in fact the resulting state representation remains holistic and is significantly easier to work with; e.g., computing the distance between different symbolic states can be handled more naturally as a part of the distance function for the overall composite state rather than independently in a discrete subspace.

5.3.3 Interactions with collision checking

Another small point of interaction in implementing TMP methods against existing motion planning libraries arises with collision checking. Although some specialized motion planning samplers make use of detailed collision detection information to bias sampling, the primary abstractions used for implementing state and motion validity checking in motion planning libraries do not expose this information or other potentially reusable computational results from collision checking. TMP also needs this information—for example, both collision checking and sampling for precondition-satisfying states starts with the result of forward kinematics, and redundantly computing these values wastes effort. Similarly, details from failed collision checks—identities of colliding objects, vectors representing the amount of penetration between object geometries—can be important for adding motion-informed constraints to the symbolic planning side of a TMP problem. Designing (potentially bespoke) subsystems for collision checking and forward kinematics with this level of information sharing and computation reuse in mind can be important for planner efficiency and effectiveness.

BIBLIOGRAPHY

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. "Using Inaccurate Models in Reinforcement Learning". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, June 25, 2006, pp. 1–8. DOI: 10/dsthn5.

[2] R. Alami, T. Siméon, and J.-P. Laumond. "A Geometrical Approach to Planning Manipulation Tasks. The case of discrete placements and grasps". In: *Proceedings of the fifth International Symposium on Robotics Research*. 1989, pp. 113–119. URL: https://hal.archives-ouvertes.fr/hal-01309950/document.

[3] T. Alamo, J. M. Bravo, and E. F. Camacho. "Guaranteed State Estimation by Zonotopes". In: *Automatica* 41.6 (June 1, 2005), pp. 1035–1043. DOI: 10/dgpn8t. URL: http://www.sciencedirect.com/science/article/pii/S0005109805000294.

[4] Matthias Althoff. "Reachability Analysis of Nonlinear Systems Using Conservative Polynomialization and Non-Convex Sets". In: *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*. Hscc '13. New York, NY, USA: Association for Computing Machinery, Apr. 8, 2013, pp. 173–182. DOI: 10/gg9djk. URL: http://doi.org/10.1145/2461328.2461358.

[5] Rajeev Alur, Thao Dang, and Franjo Ivančić1. "Counter-Example Guided Predicate Abstraction of Hybrid Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Hubert Garavel and John Hatcliff. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 208–223. DOI: 10/b5z65g.

[6] Barrett Ames, Allison Thackston, and George Konidaris. “Learning Symbolic Representations for Planning with Parameterized Skills”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 526–533. DOI: 10/gghp9q. URL: https://ieeexplore.ieee.org/abstract/document/8594313/.

[7] Jennifer Barry, Kaijen Hsiao, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. “Manipulation with Multiple Action Types”. In: *Experimental Robotics*. International Symposium on Experimental Robotics (ISER). Ed. by Jaydev P. Desai, Gregory Dudek, Oussama Khatib, and Vijay Kumar. Vol. 88. Heidelberg: Springer International Publishing, 2013, pp. 531–545. DOI: 10/ggkxd4. URL: http://link.springer.com/10.1007/978-3-319-00065-7%5C%5F36.

[8] Jennifer Barry, Leslie Pack Kaelbling, and Tomas Lozano-Perez. “A Hierarchical Approach to Manipulation with Diverse Actions”. In: 2013 IEEE International Conference on Robotics and Automation (ICRA). Karlsruhe, Germany: IEEE, May 2013, pp. 1799–1806. URL: http://ieeexplore.ieee.org/document/6630814/.

[9] Dmitry Berenson, Siddhartha S. Srinivasa, and James Kuffner. “Task Space Regions: A Framework for Pose-Constrained Manipulation Planning”. In: *International Journal of Robotics Research* 30.12 (2011), pp. 1435–1460. DOI: 10/dq4vt8. URL: https://repository.cmu.edu/cgi/viewcontent.cgi?article=2024%5C&context=robotics.

[10] Andrey Bernstein and Nahum Shimkin. “Adaptive-Resolution Reinforcement Learning with Polynomial Exploration in Deterministic Domains”. In: *Machine Learning* 81.3 (Dec. 2010), pp. 359–397. DOI: 10/d7qv5v. URL: http://link.springer.com/10.1007/s10994-010-5186-7.

[11] Julien Bidot, Bernd Schattenberg, and Susanne Biundo. “Plan Repair in Hybrid Planning”. In: *KI 2008: Advances in Artificial Intelligence*. Ed. by Andreas R. Dengel, Karsten Berns, Thomas M. Breuel, Frank Bomarius, and Thomas R. Roth-Berghofer. Vol. 5243. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 169–176. DOI: 10.1007/978-3-540-85845-4_21. URL: http://link.springer.com/10.1007/978-3-540-85845-4%5C%5F21.

[12] Avrim L Blum and Merrick L Furst. “Fast Planning Through Planning Graph Analysis”. In: *Artificial Intelligence* 90.1-2 (1997), pp. 281–300. DOI: 10/b5qhrr.

[13] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. *JAX: Composable Transformations of Python+NumPy Programs*. Version 0.2.5. 2018. URL: http://github.com/google/jax.

[14] M. S. Branicky, M. M. Curtiss, J. A. Levine, and S. B. Morgan. “RRTs for Nonlinear, Discrete, and Hybrid Planning and Control”. In: 42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475). Vol. 1. Dec. 2003, 657–663 Vol.1. DOI: 10/d3xh4f.

[15] Daniel Burfoot, Joelle Pineau, and Gregory Dudek. “RRT-Plan: A Randomized Algorithm for STRIPS Planning”. In: *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling* (Cumbria, UK). ICAPS’06. Cumbria, UK: AAAI Press, 2006, pp. 362–365. URL: http://dl.acm.org/citation.cfm?id=3037104.3037155.

[16] Stéphane Cambon, Rachid Alami, and Fabien Gravot. “A Hybrid Approach to Intricate Motion, Manipulation and Task Planning”. In: *International Journal*

of Robotics Research 28.1 (Jan. 1, 2009), pp. 104–126. DOI: 10/b6c4xv. URL: http://journals.sagepub.com/doi/10.1177/0278364908097884.

[17] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. "The Pinocchio C++ Library : A Fast and Flexible Implementation of Rigid Body Dynamics Algorithms and Their Analytical Derivatives". In: 2019 IEEE/SICE International Symposium on System Integration (SII). Jan. 2019, pp. 614–619. DOI: 10/gncq9h.

[18] Glen Chou, Necmiye Ozay, and Dmitry Berenson. "Explaining Multi-Stage Tasks by Learning Temporal Logic Formulas from Suboptimal Demonstrations". In: *Robotics: Science and Systems XVI*. July 12, 2020. DOI: 10/gg44v5. URL: http://www.roboticsproceedings.org/rss16/p097.pdf.

[19] Edmund Clarke, Ansgar Fehnker, Zhi Han, Bruce Krogh, Olaf Stursberg, and Michael Theobald. "Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Hubert Garavel and John Hatcliff. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 192–207. DOI: 10/dkt3t4.

[20] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. "Counterexample-Guided Abstraction Refinement". In: *Computer Aided Verification*. Ed. by E. Allen Emerson and Aravinda Prasad Sistla. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 154–169. DOI: 10/bjp9fs.

[21] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. "Counterexample-Guided Abstraction Refinement for Symbolic Model Check-

ing". In: *Journal of the ACM (JACM)* 50.5 (Sept. 1, 2003), pp. 752–794. DOI: 10/cx67fj. URL: https://doi.org/10.1145/876638.876643.

[22] Andrew R. Conn, Nicholas I. M. Gould, and Philippe Toint. "A Globally Convergent Augmented Lagrangian Algorithm for Optimization with General Constraints and Simple Bounds". In: *SIAM Journal on Numerical Analysis* 28.2 (Apr. 1, 1991), pp. 545–572. DOI: 10/cmng6c. URL: http://epubs.siam.org/doi/10.1137/0728030.

[23] Erwin Coumans et al. "Bullet Physics Library". In: *Open Source: bullet-physics.org* 15.49 (2013), p. 5. URL: https://pybullet.org/wordpress/.

[24] Neil T. Dantam, Swarat Chaudhuri, and Lydia E. Kavraki. "The Task-Motion Kit: An Open Source, General-Purpose Task and Motion-Planning Framework". In: *IEEE Robotics & Automation Magazine* 25.3 (Sept. 2018), pp. 61–70. DOI: 10/gftxdz. URL: https://ieeexplore.ieee.org/document/8360542/.

[25] Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. "Incremental Task and Motion Planning: A Constraint-Based Approach". In: *Robotics: Science and Systems XII*. 2016. DOI: 10.15607/rss.2016.xii.002.

[26] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 337–340. DOI: 10/fj7gzb.

[27] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. "Semantic Attachments for Domain-Independent

Planning Systems". In: *Towards Service Robots for Everyday Environments*. Ed. by Erwin Prassler, Marius Zöllner, Rainer Bischoff, Wolfram Burgard, Robert Haschke, Martin Hägele, Gisbert Lawitzky, Bernhard Nebel, Paul Plöger, and Ulrich Reiser. Vol. 76. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 99–115. DOI: 10.1007/978-3-642-25116-0_9. URL: http://link.springer.com/10.1007/978-3-642-25116-0%5C%5F9.

[28] Brian Drabble, Je Dalton, and Austin Tate. "Repairing Plans On-the-Fly". In: *Proceedings of the Nasa Workshop on Planning and Scheduling for Space*. 1997.

[29] Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras. "Combining High-Level Causal Reasoning with Low-Level Geometric Reasoning and Motion Planning for Robotic Manipulation". In: 2011 IEEE International Conference on Robotics and Automation. May 2011, pp. 4575–4581. DOI: 10/dpbbq8.

[30] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. "Temporal Logic Motion Planning for Mobile Robots". In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation. Apr. 2005, pp. 2020–2025. DOI: 10/brh68p.

[31] Kuan Fang, Yuke Zhu, Animesh Garg, Andrey Kurenkov, Viraj Mehta, Li Fei-Fei, and Silvio Savarese. *Learning Task-Oriented Grasping for Tool Manipulation from Simulated Self-Supervision*. June 24, 2018. arXiv: 1806.09266 [cs, stat]. URL: http://arxiv.org/abs/1806.09266.

[32] Richard E. Fikes and Nils J. Nilsson. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving". In: *Artificial Intelligence* 2.3-4 (Dec. 1971), pp. 189–208. DOI: 10/dvvqhw. URL: http://linkinghub.elsevier.com/retrieve/pii/0004370271900105.

[33] Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. “Plan Stability: Replanning versus Plan Repair”. In: *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*. ICAPS’06. Cumbria, UK: AAAI Press, 2006, pp. 212–221.

[34] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. “Batch Informed Trees (BIT*): Sampling-Based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)* (May 2015), pp. 3067–3074. DOI: 10/ghqdw9. arXiv: 1405.5848. URL: http://arxiv.org/abs/1405.5848.

[35] Jonathan D. Gammell and Marlin P. Strub. “Asymptotically Optimal Sampling-Based Motion Planning Methods”. In: *Annu. Rev. Control Robot. Auton. Syst.* 4.1 (May 3, 2021), pp. 295–318. DOI: 10/gjz9kp. arXiv: 2009.10484. URL: http://arxiv.org/abs/2009.10484.

[36] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. *Integrated Task and Motion Planning*. Oct. 2, 2020. arXiv: 2010.01083 [cs]. URL: http://arxiv.org/abs/2010.01083.

[37] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. “FFRob: An Efficient Heuristic for Task and Motion Planning”. In: *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*. Springer Tracts in Advanced Robotics. Springer, Cham, 2014, pp. 179–195. DOI: 10.1007/978-3-319-16595-0_11.

[38] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. “FFRob: Leveraging Symbolic Planning for Efficient Task and Motion Planning”. In: *The International Journal of Robotics Research* (2017), p. 027836491773911. DOI:

10/gcrnrp. arXiv: 1608.01335. URL: http://journals.sagepub.com/doi/10.1177/0278364917739114.

[39] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. *PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning*. Mar. 23, 2020. arXiv: 1802.08705 [cs]. URL: http://arxiv.org/abs/1802.08705.

[40] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. *Sampling-Based Methods for Factored Task and Motion Planning*. Jan. 2, 2018. arXiv: 1508.04886v1. URL: http://arxiv.org/abs/1801.00680.

[41] Caelan Reed Garrett, Chris Paxton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Dieter Fox. *Online Replanning in Belief Space for Partially Observable Task and Motion Problems*. Nov. 11, 2019. arXiv: 1911.04577 [cs]. URL: http://arxiv.org/abs/1911.04577.

[42] Ilche Georgievski and Marco Aiello. *An Overview of Hierarchical Task Network Planning*. Mar. 28, 2014. arXiv: 1403.7426 [cs]. URL: http://arxiv.org/abs/1403.7426.

[43] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge: Cambridge University Press, 2014. DOI: 10.1017/cbo9781139583923. URL: http://ebooks.cambridge.org/ref/id/CBO9781139583923.

[44] Robert L Grossman, Anil Nerode, Anders P Ravn, and Hans Rischel. *Hybrid Systems*. Vol. 736. Springer, 1993.

[45] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107. DOI: 10/c9zgc4.

[46] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool, 2019. URL: https://ieeexplore.ieee.org/document/8681776.

[47] Kris Hauser and Jean-Claude Latombe. "Integrating Task and PRM Motion Planning: Dealing with Many Infeasible Motion Planning Queries". In: *ICAPS 2009 Workshop on Bridging the Gap between Task and Motion Planning*. 2009.

[48] Kris Hauser and Jean-Claude Latombe. "Multi-Modal Motion Planning in Non-Expansive Spaces". In: *The International Journal of Robotics Research* 29.7 (June 2010), pp. 897–915. DOI: 10/c65pps. URL: http://journals.sagepub.com/doi/10.1177/0278364909352098.

[49] Kris Hauser and Victor Ng-Thow-Hing. "Randomized Multi-Modal Motion Planning for a Humanoid Robot Manipulation Task". In: *The International Journal of Robotics Research* 30.6 (May 2011), pp. 678–698. DOI: 10/cf2fb6. URL: http://journals.sagepub.com/doi/10.1177/0278364910386985.

[50] M. Helmert. "The Fast Downward Planning System". In: *Journal of Artificial Intelligence Research* 26 (July 12, 2006), pp. 191–246. DOI: 10/gg4fh7. URL: https://jair.org/index.php/jair/article/view/10457.

[51] Jörg Hoffmann and Bernhard Nebel. "The FF Planning System: Fast Plan Generation through Heuristic Search". In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 263–312. DOI: 10/gcp3qj. arXiv: 1106.0675. URL: https://arxiv.org/abs/1106.0675.

[52] Nicholas K. Jong and Peter Stone. "Model-Based Function Approximation in Reinforcement Learning". In: *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '07. New York,

NY, USA: Association for Computing Machinery, May 14, 2007, pp. 1–8. DOI: 10/ch6cjm.

[53] Leslie Pack Kaelbling and Tomás Lozano-Pérez. "Integrated Task and Motion Planning in Belief Space". In: *International Journal of Robotics Research* 32.9-10 (2013), pp. 1194–1227. DOI: 10/f5fg4c.

[54] Leslie Pack Kaelbling and Tomás Lozano-Pérez Tomas. "Hierarchical Task and Motion Planning in the Now". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1470–1477. DOI: 10.1109/icra.2011.5980391. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5980391.

[55] Sertac Karaman and Emilio Frazzoli. "Sampling-Based Algorithms for Optimal Motion Planning". In: *The International Journal of Robotics Research* 30.7 (June 2011), pp. 846–894. DOI: 10/c2wgw5. URL: http://journals.sagepub.com/doi/10.1177/0278364911406761.

[56] Erez Karpas and Daniele Magazzeni. "Automated Planning for Robotics". In: *Annu. Rev. Control Robot. Auton. Syst.* 3.1 (May 3, 2020), pp. 417–439. DOI: 10/ggtt8v. URL: https://www.annualreviews.org/doi/10.1146/annurev-control-082619-100135.

[57] Henry Kautz, David McAllester, and Bart Selman. "Encoding Plans in Propositional Logic". In: International Conference on Principles of Knowledge Representation and Reasoning. 1996, p. 11. URL: https://www.cs.cornell.edu/selman/papers/pdf/96.kr.plan.pdf.

[58] Henry Kautz and Bart Selman. "Unifying SAT-Based and Graph-Based Planning". In: International Joint Conferences on Artificial Intelligence. Vol. 99.

1999, pp. 318–325. URL: https://www.cs.cornell.edu/selman/papers/pdf/99.ijcai.blackbox.pdf.

[59] Henry A Kautz, Bart Selman, et al. “Planning as Satisfiability”. In: *European Conference on Artificial Intelligence*. Vol. 92. 1992, pp. 359–363.

[60] L. E. Kavraki, P. Svestka, J.- Latombe, and M. H. Overmars. “Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces”. In: *IEEE Transactions on Robotics and Automation* 12.4 (Aug. 1996), pp. 566–580. DOI: 10/fsgth3.

[61] Beomjoon Kim, Leslie Pack Kaelbling, and Tomas Lozano-Perez. *Guiding the Search in Continuous State-Action Spaces by Learning an Action Sampling Distribution from off-Target Samples*. Nov. 4, 2017. arXiv: 1711.01391 [cs]. URL: http://arxiv.org/abs/1711.01391.

[62] Beomjoon Kim, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. “Adversarial Actor-Critic Method for Task and Motion Planning Problems Using Planning Experience”. In: *AAAI* 33 (July 17, 2019), pp. 8017–8024. DOI: 10/gghffm. URL: https://aaai.org/ojs/index.php/AAAI/article/view/4803.

[63] Zachary Kingston, Mark Moll, and Lydia E Kavraki. “Decoupling Constraints from Sampling-Based Planners”. In: *International Symposium of Robotics Research*. 2017, pp. 1–16.

[64] Zachary Kingston, Mark Moll, and Lydia E. Kavraki. “Sampling-Based Methods for Motion Planning with Constraints”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 1.1 (2018), pp. 159–185. DOI: 10/gf5zq4.

[65] Zachary Kingston, Andrew M Wells, Mark Moll, and Lydia E Kavraki. "Informing Multi-Modal Planning with Synergistic Discrete Leads". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, p. 7.

[66] Niklas Kochdumper and Matthias Althoff. *Constrained Polynomial Zonotopes*. May 18, 2020. arXiv: 2005.08849 [math]. URL: http://arxiv.org/abs/2005.08849.

[67] Niklas Kochdumper and Matthias Althoff. *Sparse Polynomial Zonotopes: A Novel Set Representation for Reachability Analysis*. Jan. 7, 2019. arXiv: 1901.01780 [cs]. URL: http://arxiv.org/abs/1901.01780.

[68] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. "From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning". In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 215–289. DOI: 10/gdtnhb. URL: http://www.jair.org/papers/paper5575.html.

[69] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. "Temporal-Logic-Based Reactive Mission and Motion Planning". In: *IEEE Transactions on Robotics* 25.6 (Dec. 2009), pp. 1370–1381. DOI: 10/bhzcss.

[70] Hadas Kress-Gazit, Tichakorn Wongpiromsarn, and Ufuk Topcu. "Correct, Reactive, High-Level Robot Control". In: *IEEE Robotics Automation Magazine* 18.3 (Sept. 2011), pp. 65–74. DOI: 10/b4k7kw.

[71] Fabien Lagriffoul, Neil T. Dantam, Caelan Garrett, Aliakbar Akbari, Siddharth Srivastava, and Lydia E. Kavraki. "Platform-Independent Benchmarks for Task and Motion Planning". In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 3765–3772. DOI: 10/gd4vzv. URL: https://ieeexplore.ieee.org/document/8411475/.

[72] Fabien Lagriffoul, Dimitar Dimitrov, Julien Bidot, Alessandro Saffiotti, and Lars Karlsson. “Efficiently Combining Task and Motion Planning Using Geometric Constraints”. In: *The International Journal of Robotics Research* 33.14 (2014), pp. 1726–1747. DOI: 10/f25hfj.

[73] Fabien Lagriffoul, Dimitar Dimitrov, Alessandro Saffiotti, and Lars Karlsson. “Constraint Propagation on Interval Bounds for Dealing with Geometric Backtracking”. In: *IEEE International Conference on Intelligent Robots and Systems*. 2012, pp. 957–964. DOI: 10/f3nsfp.

[74] Fabien Lagriffoul, Lars Karlsson, Julien Bidot, and Alessandro Saffiotti. “Combining Task and Motion Planning Is Not Always a Good Idea”. In: *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*. 2013.

[75] Steven M. LaValle. “From Dynamic Programming to RRTs: Algorithmic Design of Feasible Trajectories”. In: *Control Problems in Robotics*. Ed. by Antonio Bicchi, Domenico Prattichizzo, and Henrik Iskov Christensen. Vol. 4. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 19–37. DOI: 10.1007/3-540-36224-x_2. URL: http://link.springer.com/10.1007/3-540-36224-X%5C%5F2.

[76] Steven M. LaValle. *Planning Algorithms*. Cambridge: Cambridge University Press, 2006. DOI: 10.1017/cbo9780511546877. URL: https://www.cambridge.org/core/product/identifier/9780511546877/type/book.

[77] Steven M. LaValle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Tech. rep. May 1999.

[78] Vu Tuan Hieu Le, Cristina Stoica, Teodoro Alamo, Eduardo F. Camacho, and Didier Dumur. *Zonotopes: From Guaranteed State-Estimation to Control*. Hoboken, USA: John Wiley & Sons, Inc., Oct. 11, 2013. DOI: 10.1002/9781118761588.

[79] Allan M. M. Leal. *Autodiff, a Modern, Fast and Expressive C++ Library for Automatic Differentiation*. 2018. URL: https://autodiff.github.io.

[80] Weiwei Li and Emanuel Todorov. "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems." In: *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics*. Setúbal, Portugal: SciTePress - Science, 2004, pp. 222–229. DOI: 10/fn7nnp. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0001143902220229.

[81] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge, UK: Cambridge University Press, 2017. 528 pp.

[82] Masoumeh Mansouri, Federico Pecora, and Peter Schüller. "Combining Task and Motion Planning: Challenges and Guidelines". In: *Front. Robot. AI* 8 (2021). DOI: 10/gj36bc.

[83] Drew McDermott. "PDDL-the Planning Domain Definition Language". In: AIPS Planning Competition. 1998.

[84] Toki Migimatsu and Jeannette Bohg. *Object-Centric Task and Motion Planning in Dynamic Environments*. Nov. 12, 2019. arXiv: 1911.04679 [cs]. URL: http://arxiv.org/abs/1911.04679.

[85] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research. New York: Springer, 2006. 664 pp.

[86] Michael Noseworthy, Rohan Paul, Subhro Roy, Daehyung Park, and Nicholas Roy. “Task-Conditioned Variational Autoencoders for Learning Movement Primitives”. In: *Conference on Robot Learning*. Conference on Robot Learning. Proceedings of Machine Learning Research, May 12, 2020, pp. 933–944. URL: http://proceedings.mlr.press/v100/noseworthy20a.html.

[87] Ali Nouri and Michael L. Littman. “Multi-Resolution Exploration in Continuous Spaces”. In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou. Curran Associates, Inc., 2009, pp. 1209–1216. URL: http://papers.nips.cc/paper/3557-multi-resolution-exploration-in-continuous-spaces.pdf.

[88] Stanley Osher and Ronald Fedkiw. “Signed Distance Functions”. In: *Level Set Methods and Dynamic Implicit Surfaces*. Ed. by Stanley Osher and Ronald Fedkiw. Applied Mathematical Sciences. New York, NY: Springer, 2003, pp. 17–22. DOI: 10.1007/0-387-22746-6_2.

[89] Adam Pacheck, Salar Moarref, and Hadas Kress-Gazit. “Finding Missing Skills for High-Level Behaviors”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 10335–10341. DOI: 10/gmjj58.

[90] Erion Plaku and Gregory D. Hager. “Sampling-Based Motion and Symbolic Action Planning with Geometric and Differential Constraints”. In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2010, pp. 5002–5008. DOI: 10.1109/robot.2010.5509563. URL: http://ieeexplore.ieee.org/abstract/document/5509563/.

[91] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. “Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning”. In: Robotics: Science and Systems. 2007, p. 8. DOI: 10/gnbt52. URL: http://www.roboticsproceedings.org/rss03/p40.pdf.

[92] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. "Motion Planning With Dynamics by a Synergistic Combination of Layers of Planning". In: *IEEE Trans. Robot.* 26.3 (June 2010), pp. 469–482. DOI: 10/dtztj3. URL: http://ieeexplore.ieee.org/document/5477241/.

[93] Vignesh Raghuraman and Justin P. Koeln. *Set Operations and Order Reductions for Constrained Zonotopes*. Sept. 13, 2020. arXiv: 2009.06039 [cs, eess]. URL: http://arxiv.org/abs/2009.06039.

[94] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. *Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations*. June 26, 2018. arXiv: 1709.10087 [cs]. URL: http://arxiv.org/abs/1709.10087.

[95] Divyam Rastogi, Ivan Koryakovskiy, and Jens Kober. "Sample-Efficient Reinforcement Learning via Difference Models". In: Third Machine Learning in Planning and Control of Robot Motion Workshop at ICRA. 2018, p. 6.

[96] Harish Ravichandar, S. Reza Ahmadzadeh, M. Asif Rana, and Sonia Chernova. *Skill Acquisition via Automated Multi-Coordinate Cost Balancing*. Mar. 27, 2019. arXiv: 1903.11725 [cs]. URL: http://arxiv.org/abs/1903.11725.

[97] Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. "Recent Advances in Robot Learning from Demonstration". In: *Annu. Rev. Control Robot. Auton. Syst.* 3.1 (May 3, 2020), pp. 297–330. DOI: 10/ghjknd. URL: https://www.annualreviews.org/doi/10.1146/annurev-control-100819-063206.

[98] Tianyu Ren, Georgia Chalvatzaki, and Jan Peters. *Extended Task and Motion Planning of Long-Horizon Robot Manipulation*. Mar. 9, 2021. arXiv: 2103.05456 [cs]. URL: http://arxiv.org/abs/2103.05456.

[99] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard. "Optimal, Sampling-Based Manipulation Planning". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. May 2017, pp. 3426–3432. DOI: 10/gf5zq5.

[100] John Schulman, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. "Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization". In: *Robotics: Science and Systems IX*. Robotics: Science and Systems Foundation, June 23, 2013. DOI: 10/gg55gd. URL: http://www.roboticsproceedings.org/rss09/p31.pdf.

[101] Joseph K. Scott, Davide M. Raimondo, Giuseppe Roberto Marseglia, and Richard D. Braatz. "Constrained Zonotopes: A New Tool for Set-Based Estimation and Fault Detection". In: *Automatica* 69 (July 1, 2016), pp. 126–136. DOI: 10/f8qwf5. URL: http://www.sciencedirect.com/science/article/pii/S0005109816300772.

[102] Naman Shah, Deepak Kala Vasudevan, Kislay Kumar, Pranav Kamojjhala, and Siddharth Srivastava. *Anytime Integrated Task and Motion Policies for Stochastic Environments*. May 29, 2020. arXiv: 1904.13006 [cs]. URL: http://arxiv.org/abs/1904.13006.

[103] Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. "Emergent Real-World Robotic Skills via Unsupervised Off-Policy Reinforcement Learning". In: *Robotics: Science and Systems XVI*. Robotics: Science and Systems Foundation, July 12, 2020. DOI: 10/gg44vt. URL: http://www.roboticsproceedings.org/rss16/p053.pdf.

[104] Rahul Shome, Daniel Nakhimovich, and Kostas E Bekris. “Pushing the Boundaries of Asymptotic Optimality in Integrated Task and Motion Planning”. In: Workshop on the Algorithmic Foundations of Robotics. 2020, p. 16.

[105] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. “Combined Task and Motion Planning through an Extensible Planner-Independent Interface Layer”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 639–646. DOI: 10.1109/icra.2014.6906922.

[106] Marlin P. Strub and Jonathan D. Gammell. “Adaptively Informed Trees (AIT*): Fast Asymptotically Optimal Path Planning through Adaptive Heuristics”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. May 2020, pp. 3191–3198. DOI: 10/ghvk26. URL: https://arxiv.org/abs/2002.06599.

[107] Ioan A. Şucan and Lydia E. Kavraki. “Accounting for Uncertainty in Simultaneous Task and Motion Planning Using Task Motion Multigraphs”. In: *2012 IEEE International Conference on Robotics and Automation*. May 2012, pp. 4822–4828. DOI: 10/gjsrpx.

[108] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. “The Open Motion Planning Library”. In: *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012), pp. 72–82. DOI: 10/gdnbz5. URL: http://ompl.kavrakilab.org.

[109] Richard S. Sutton. “Dyna, an Integrated Architecture for Learning, Planning, and Reacting”. In: *SIGART Bull.* 2.4 (July 1, 1991), pp. 160–163. DOI: 10/cd8s9h. URL: http://doi.org/10.1145/122344.122377.

[110] Wil Thomason and Ross A Knepper. “A Unified Sampling-Based Approach to Integrated Task and Motion Planning”. In: *International Symposium on Robotics*

Research (ISRR). 2019. URL: https://www.cs.cornell.edu/~wil/papers/isrr2019%5C%5Funifiedtamp.pdf.

[111] Wil Thomason and Hadas Kress-Gazit. *Counterexample-Guided Repair for Symbolic-Geometric Action Abstractions*. May 13, 2021. arXiv: 2105.06537 [cs]. URL: http://arxiv.org/abs/2105.06537.

[112] Marc Toussaint. "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning". In: International Joint Conference on Artificial Intelligence. 2015, p. 7. URL: http://ijcai.org/Proceedings/15/Papers/274.pdf.

[113] Marc Toussaint, Kelsey Allen, Kevin Smith, and Joshua Tenenbaum. "Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning". In: *Robotics: Science and Systems XIV*. Robotics: Science and Systems Foundation, June 26, 2018. DOI: 10/gf66q4. URL: http://www.roboticsproceedings.org/rss14/p44.pdf.

[114] Roman Van Der Krogt and Mathijs De Weerdt. "Plan Repair as an Extension of Planning." In: *ICAPS*. Vol. 5. 2005, pp. 161–170.

[115] Cristian-Ioan Vasile, Vasumathi Raman, and Sertac Karaman. "Sampling-Based Synthesis of Maximally-Satisfying Controllers for Temporal Logic Specifications". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3840–3847. DOI: 10/ggfqms.

[116] William Vega-Brown and Nicholas Roy. "Task and Motion Planning Is PSPACE-Complete". In: AAAI. 2020, p. 8. DOI: 10/gg87db. URL: https://aaai.org/Papers/AAAI/2020GB/AAAI-Vega-brownW.10305.pdf.

[117] Anirudh Vemula, Yash Oza, J. Bagnell, and Maxim Likhachev. "Planning and Execution Using Inaccurate Models with Provable Guarantees". In: *Proceedings*

of Robotics: Science and Systems. Robotics: Science and Systems. Corvalis, Oregon, USA, July 2020. DOI: 10.15607/rss.2020.xvi.001. URL: http://roboticsproceedings.org/rss16/p001.html.

[118] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. "Active Model Learning and Diverse Action Sampling for Task and Motion Planning". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 4107–4114. DOI: 10/gghb4s.

[119] Zi Wang, Caelan Reed Garrett, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. *Learning Compositional Models of Robot Skills for Task and Motion Planning*. June 8, 2020. arXiv: 2006.06444 [cs]. URL: http://arxiv.org/abs/2006.06444.

[120] Andrew M Wells, Zachary Kingston, Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. "Finite-Horizon Synthesis for Probabilistic Manipulation Domains". In: ICRA. 2021, p. 7.

[121] Victoria Xia, Zi Wang, and Leslie Pack Kaelbling. *Learning Sparse Relational Transition Models*. Oct. 25, 2018. arXiv: 1810.11177 [cs, stat]. URL: http://arxiv.org/abs/1810.11177.

[122] Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. "CHOMP: Covariant Hamiltonian Optimization for Motion Planning". In: *The International Journal of Robotics Research* 32.9-10 (Aug. 1, 2013), pp. 1164–1193. DOI: 10/f5fhn9. URL: https://doi.org/10.1177/0278364913488805.