Article

The International Journal of
Robotics Research
2018, Vol. 37(13-14) 1796–1825
© The Author(s) 2018
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/0278364918802962
journals.sagepub.com/home/ijr

Sampling-based methods for factored task and motion planning

Caelan Reed Garrett, Tomás Lozano-Pérez and Leslie Pack Kaelbling

Abstract

This paper presents a general-purpose formulation of a large class of discrete-time planning problems, with hybrid state and control-spaces, as factored transition systems. Factoring allows state transitions to be described as the intersection of several constraints each affecting a subset of the state and control variables. Robotic manipulation problems with many movable objects involve constraints that only affect several variables at a time and therefore exhibit large amounts of factoring. We develop a theoretical framework for solving factored transition systems with sampling-based algorithms. The framework characterizes conditions on the submanifold in which solutions lie, leading to a characterization of robust feasibility that incorporates dimensionality-reducing constraints. It then connects those conditions to corresponding conditional samplers that can be composed to produce values on this submanifold. We present two domain-independent, probabilistically complete planning algorithms that take, as input, a set of conditional samplers. We demonstrate the empirical efficiency of these algorithms on a set of challenging task and motion planning problems involving picking, placing, and pushing.

Keywords
task and motion planning, manipulation planning, AI reasoning

1. Introduction

Many important robotic applications require planning in a high-dimensional space that includes not just the robot configuration, but also the “configuration” of the external world, including poses and attributes of objects. There has been a great deal of progress in developing probabilistically complete sampling-based methods that move beyond motion planning to these hybrid problems including various forms of task planning. These new methods each require a new formulation, theoretical framework, sampling method, and search algorithm. We propose a general-purpose abstraction of sampling-based planning for a class of hybrid systems that implements each of these requirements. As our motivating application, we apply this abstraction to robot task-and-motion planning.

We model planning using *factored transition systems*, discrete-time planning problems involving mixed discrete-continuous state and control spaces. This formulation is able to highlight any *factoring* present within the problem resulting from constraints that only impact a few variables at a time. Directly exposing factoring enables to us to design algorithms that are able to efficiently sample states and controls by sampling values for subsets of the variables at a time. In addition, factoring enables the use of efficient discrete search algorithms from artificial intelligence planning community.

The theoretical contribution of this paper is an analysis of the topology of a problem’s solution space, particularly in the presence of dimensionality-reducing constraints. The key insight is that, in some cases, the intersection of several lower-dimensional constraints lies on a submanifold of the parameter space that can be identified using only the individual constraints. By understanding the topology of the solution space, we define a property that characterizes a large class of problems for which sampling-based planning methods can be successful.

The algorithmic contribution is the construction of two sampling-based planning algorithms that exploit the factored, compositional structure of the solution space to draw samples using *conditional samplers*. These algorithms search in a combined space that includes the discrete structure (which high-level operations, such as “pick” or “place”

MIT CSAIL, Cambridge, MA, USA

Corresponding author:
Caelan Reed Garrett, Computer Science and Artificial Intelligence Laboratory, MIT, 32 Vassar Street, Cambridge, MA 02139, USA.
Email: caelan@csail.mit.edu

happen in which order) and parameters (particular continuous parameters of the actions) of a solution. Theoretically, these algorithms are probabilistically complete when given a set of sampling primitives that cover the appropriate spaces with probability one. Practically, they can solve complex instances of task-and-motion planning problems as well as problems involving non-prehensile manipulation problems.

2. Related work

Planning problems in which the goal is not just to move the robot without collision but also to operate on the objects in the world have been addressed from the earliest days of manipulation planning (Lozano-Pérez, 1981; Lozano-Pérez et al., 1987; Wilfong, 1988). Alami et al. (1994, 1990) decomposed manipulation problems using a *manipulation graph* that represents connected components of the configuration space given by a particular robot grasp. They observed that solutions are alternating sequences of *transit* and *transfer* paths corresponding to the robot moving while its hand is empty and the robot moving while holding an object. Vendittelli et al. (2015) proved a manipulation planning with a robot and two movable obstacles is decidable by providing a complete decomposition-based algorithm. Deshpande et al. (2016) extended this result to general, prehensile task and motion planning. Maintaining an explicit characterization of the free configuration space can be prohibitively expensive in high-dimensional planning problems.

Siméon et al. (2004) extended the work of Alami et al. (1994, 1990) by using probabilistic roadmaps (PRMs) (Kavraki et al., 1996) to approximate each component of the configuration space. The aSyMov system (Cambon et al., 2009) generalizes this strategy to task and motion planning with significant discrete structure. It uses a heuristic computed by planning at just the symbolic level to guide the geometric exploration of its roadmaps. Plaku and Hager (2010) also used symbolic planning to influence geometric planning but for biasing sampling instead of guiding the search.

Stilman and Kuffner (2006); Stilman et al. (2007) introduced the problem of robotic *navigation among movable obstacles* (NAMO), robotic motion planning in a reconfigurable environment. They provided an algorithm for solving monotonic problem instances, problems that require moving each object at most once. Van Den Berg et al. (2009) developed a probabilistically complete algorithm for robustly feasible NAMO problems. Krontiris and Bekris (2015, 2016) extended the work of Stilman and Kuffner (2006); Stilman et al. (2007) to non-monotonic rearrangement problems, which require moving a set of objects to specified goals poses, by using their algorithm as a primitive within a larger, complete search. These algorithms are each specialized to a subclass of manipulation planning.

Hauser and Ng-Thow-Hing (2011) introduced a framework and algorithm *multi-modal motion planning*, motion planning in overlapping spaces of non-uniform dimensionality. Barry et al. (2013); Barry (2013) considered multi-modal motion planning using bidirectional rapidly-exploring random trees (RRT-Connect). Vega-Brown and Roy (2016) extended these ideas to optimal planning with differential constraints. Because these approaches do not exploit any factoring present within a problem, they must sample entire states at once and are unable to take advantage of powerful heuristics to guide their search.

Dornhege et al. (2009, 2013) introduced *semantic attachments*, external predicates evaluated on a geometric representation of the state, to integrate geometric reasoning into artificial intelligence planners. Their algorithms assume a finite set of primitive actions that restricts them to discrete control spaces. They evaluate semantic attachments within their search, which results in unnecessarily computing many expensive motion plans.

Kaelbling and Lozano-Pérez (2011) introduced *generators* to select predecessor states in a goal regression search (HPN). Garrett et al. (2015) gave an algorithm (HBF) for planning in hybrid spaces by using approximations of the planning problem to guide the backward generation of successor actions to be considered in a forward search. Both approaches requires that generators are specified according to an inverse model to be compatible with their backward searches. In addition, both approaches integrate search and sampling preventing them from leveraging discrete search algorithms as blackbox subroutines.

Pandey et al. (2012) and de Silva et al. (2013) use *hierarchical task networks* (HTNs) (Erol et al., 1994) to guide a search over *plan skeletons*, discrete sequences of actions with unbound continuous variables, using knowledge about the task decomposition. The search over plan skeletons backtracks in the event that it is unable to bind the free continuous variables of a skeleton. Lozano-Pérez and Kaelbling (2014) took a similar approach but leverage constraint satisfaction problem (CSP) solvers operating on discretized variable domains to bind the free variables. Lagriffoul et al. (2014, 2012) also searched over plan skeletons but were able to prune some plan skeletons from consideration using computed bounds on the constraints. For each plan skeleton under consideration, they generated a set of approximate linear constraints, e.g. from grasp and placement choices, and used linear programming to compute a valid assignment of continuous values or determine that one does not exist. Similarly, Toussaint (2015); Toussaint and Lopes (2017) formulated the binding of geometric variables as a nonlinear constrained optimization problem and use a hierarchy of bounds on the nonlinear program to prune plan skeletons. Because binding and pruning operate globally on entire plan skeletons, these approaches are unable to identify individual variables and constraints that primarily contributed to infeasibility. Thus, the search over plan skeletons may evaluate many similar plan skeletons that exhibit the same behavior. In contrast, by reasoning locally about individual conditional samplers, our focused algorithm is able to retain samples that satisfy their associated constraints and focus the subsequent search and sampling on conditional samplers that failed to produce satisfactory samples.

The FFRob algorithm of Garrett et al. (2014) samples a set of object poses and robot configurations and then plans with them using a search algorithm that incorporates geometric constraints in its heuristic. An iterative version of FFRob that repeats this process until a solution is found is probabilistically complete and exponentially convergent (Garrett et al., 2017a). Our incremental algorithm can be seen as generalizing this strategy of iteratively sampling then searching from pick-and-place domains to domains with arbitrary conditional samplers.

Erdem et al. (2011) planned at the task-level using a Boolean satisfiability (SAT) solver, initially ignoring geometric constraints, and then attempt to produce motion plans satisfying the task-level actions. If an induced motion planning problem is infeasible, the task-level description is updated to indicate motion infeasibility using a domain-specific diagnostic interface. Dantam et al. (2016) extended this work by using a satisfiability modulo theories (SMT) solver to incrementally add new constraints and restart the task-level search from its previous state. Their approach also adjusts to motion planning failures automatically and in a way that allows previously failed motion planning queries to be reconsidered. The algorithms of Erdem et al. (2011) and Dantam et al. (2016) both assume an a priori discretization of all continuous values apart from configurations, for example, objects placements. Srivastava et al. (2014) removed this restriction by using symbolic references to continuous parameters. Upon finding a task-level plan, they use a domain-specific interface, like Erdem et al. (2011), to bind values for symbolic references and update the task-level description when none are available. Our focused algorithm is related to these approaches in that it lazily evaluates constraints and plans with lazy samples before real values. However, it is able to automatically manage its search and sampling in a probabilistically complete manner using each individual conditional sampler as a blackbox, without the need for a domain-specific interface.

Our work captures many of the insights in these previous approaches in a general-purpose framework. It also highlights the role of factoring in developing efficient algorithms for sampling relevant values and searching discretized spaces.

3. Factored transition system

We begin by defining a general class of models for deterministic, observable, discrete-time, hybrid systems. These systems are *hybrid* in that they have mixed discrete-continuous state and control spaces. However, they are also *discrete-time* meaning that transitions are discrete changes to the hybrid state (Torrisi and Bemporad, 2001). This is in contrast to a *continuous-time* hybrid systems (Alur et al., 1995, 2000; Henzinger, 2000), which allow continuous transitions described as differential equations. It is possible to address many continuous-time problems in this framework, as long as they can be solved with a finite sequence of continuous control inputs.

Definition 1. A discrete-time *transition system* $\mathcal{S} = \langle \mathcal{X}, \mathcal{U}, \mathcal{T} \rangle$ is defined by a set of states (*state space*) $\mathcal{X}$, set of controls (*control space*) $\mathcal{U}$, and a *transition relation* $\mathcal{T} \subseteq \mathcal{X} \times \mathcal{U} \times \mathcal{X}$.

For many physical systems, $\mathcal{T}$ is a *transition function* from $\mathcal{X} \times \mathcal{U}$ to $\mathcal{X}$. We are interested in controlling a transition system to move from a given initial state to a state within a goal set.

Definition 2. A *problem* $\mathcal{P} = \langle x^0, X^*, \mathcal{S} \rangle$ is an initial state $x^0 \subseteq \mathcal{X}$, a set of goal states $X^* \subseteq \mathcal{X}$, and a transition system $\mathcal{S}$.

Definition 3. A *plan* for a problem $\mathcal{P}$ is finite sequence of k control inputs $(u^1, \ldots, u^k)$ and k states $(x^1, \ldots, x^k)$ such that $(x^{i-1}, u^i, x^i) \in \mathcal{T}$ for $i \in \{1, \ldots, k\}$ and $x^k \in X^*$.

When $\mathcal{T}$ is a transition function, a plan can be uniquely identified by its sequence of control inputs.

3.1. Factoring

We are particularly interested in transition systems that are factorable. As we show in Sections 6 and 8, factored structure is useful for developing efficient methods for sampling and searching transition systems.

Definition 4. A *factored transition system* is a transition system with state space $\bar{\mathcal{X}} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_m$ and control space $\bar{\mathcal{U}} = \mathcal{U}_1 \times \cdots \times \mathcal{U}_n$ that is defined by m *state variables* $\bar{x} = (x_1, \ldots, x_m)$ and n *control variables* $\bar{u} = (u_1, \ldots, u_n)$.

The transition relation is a subset of the *transition parameter space* $\mathcal{T} \subseteq \bar{\mathcal{X}} \times \bar{\mathcal{U}} \times \bar{\mathcal{X}}$ Valid transitions are $(x_1, \ldots, x_m, u_1, \ldots, u_n, x'_1, \ldots, x'_m) \in \mathcal{T}$. To simplify notation, we will generically refer to each x_p, u_p, or x'_p in a transition as a *parameter* z_p where $p \in \{1, \ldots, 2m+n\} = \Theta$ indexes the entire sequence of variables. For a subset of parameter indices $P = (p_1, \ldots, p_k) \subseteq \Theta$, let $\bar{z}_P = (z_{p_1}, \ldots, z_{p_k}) \in \bar{\mathcal{Z}}_P$ be the combined values and $\bar{\mathcal{Z}}_P = \mathcal{Z}_{p_1} \times \cdots \times \mathcal{Z}_{p_k}$ be the combined domain of the parameters.

Many transition relations are hybrid, in that there is a discrete choice between different types of operation, each of which has a different continuous constraint on the relevant parameters. For example, a pick-and-place problem has transitions corresponding to a robot moving its base, picking each object, and placing each object. To expose the discrete structure, we decompose the transition relation $\mathcal{T} = \bigcup_{a=1}^{\alpha} T_a$ into the union of α smaller *transition components* T_a. A transition relation T_a often is the intersection of several constraints on a subset of the transition parameters.

Definition 5. A *constraint* is a pair $C = \langle P, R \rangle$ where $P \subseteq \Theta$ is a subset of parameter indices and $R \subseteq \bar{\mathcal{Z}}_P$ is a relation on sets $\mathcal{Z}_{p_1}, \dots, \mathcal{Z}_{p_k}$.

A tuple of values that satisfy a constraint is called a constraint element.

Definition 6. A constraint *element* $C(v_{p_1}, \dots, v_{p_k})$ is composed of a constraint $C = \langle P, R \rangle$ and variable values $(v_{p_1}, \dots, v_{p_k}) = \bar{v}_P \in R$ for parameter indices $P = (p_1, \dots, p_k)$.

For instance, *pick* transitions involve constraints that the end-effector initially is empty, the target object is placed stably, the robot's configuration forms a kinematic solution with the placement, and the end-effector ultimately is holding the object. A constraint decomposition is particularly useful when $|P| \ll 2m + n$; i.e. each individual constraint has low arity. Let $C{\uparrow}^{\Theta} = \{\bar{z} \in \bar{\mathcal{Z}}_{\Theta} | \bar{z}_P \in R\}$ be the *extended form* of the constraint over all parameter indices Θ. This alternatively can be seen as a Cartesian product of R with $\bar{\mathcal{Z}}_{\Theta \setminus P}$ followed by permuting the parameter indices to be in a sorted order.

Definition 7. A transition component T_a is specified as the intersection over a *clause* of β constraints $\mathcal{C}_a = \{C_1, \dots, C_\beta\}$ where $T_a = \bigcap_{b=1}^{\beta} C_b^a{\uparrow}^{\Theta}$.

A clause is analogous to a conjunctive clause from Membership in T_a is equivalent to the *conjunction* over membership for each $\mathcal{C}_a$: $[\bar{z} \in T_a] \Leftarrow \wedge_{b=1}^{\beta}[\bar{z} \in C_b^a{\uparrow}^{\Theta}]$. Within a clause, there are implicit variable domain constraints on each parameter z_p of the form $Var_p = \langle (z_p), \mathcal{Z}_p \rangle$. Finally, the transition relation $\mathcal{T}$ is the union of α clauses $\{\mathcal{C}_1, \dots, \mathcal{C}_\alpha\}$. Membership in $\mathcal{T}$ is equivalent to the *disjunction* over membership for each $\mathcal{T}$: $[\bar{z} \in \mathcal{T}] \Leftarrow \vee_{a=1}^{\alpha}[\bar{z} \in T_a]$. Thus, membership in $\mathcal{T}$ is a logical expression in *disjunctive normal form* over literals $[\bar{z} \in C_b^a{\uparrow}^{\Theta}]$.

Factoring the transition relation can expose constraints that have a simple equality form. Equality constraints are important because they transparently reduce the dimensionality of the transition parameter space.

Definition 8. A *constant equality* constraint $\langle (z_p), \{\kappa\} \rangle$ (denoted $z_p = \kappa$) indicates that parameter z_p has value κ.

Definition 9. A *pairwise equality* constraint $\langle (z_p, z_{p'}), \{(v, v) | v \in \mathcal{Z}_p \cap \mathcal{Z}_{p'}\} \rangle$ (denoted $z_p = z_{p'}$) indicates that parameters $z_p, z_{p'}$ have the same value.

For many high-dimensional systems, the transition relation is *sparse*, meaning its transitions only alter a small number of the state variables at a time. Sparse transition systems have transition relations where each clause contains pairwise equality constraints $x_p = x'_p$ for most state variables p. Intuitively, most transitions do not change most state variables.

The initial state $\bar{x}^0$ and set of goal states $\bar{X}^*$ can be specified using clauses $\mathcal{C}_0$ and $\mathcal{C}_*$ defined solely on state variables. Because $\bar{x}^0$ is a single state, its clause is composed of constant equality constraints.

3.2. Constraint satisfaction

Planning can be thought of as a combined search over discrete clauses and hybrid parameter values. To select a type is to select a clause from the transition relation; to select its parameters is to select the $\bar{x}, \bar{u}, \bar{x}'$ values.

Definition 10. A finite sequence of transition components $\vec{a} = (a_1, \dots, a_k)$ is a *plan skeleton* (Lozano-Pérez and Kaelbling, 2014).

For example, solutions to pick-and-place problems are sequences of *move, pick, move-while-holding*, and *place* clauses involving the same object.

Definition 11. The *plan parameter space* for a plan skeleton $\vec{a} = (a_1, \dots, a_k)$ is an alternating sequence of states and controls $(\bar{x}^0, \bar{u}^1, \bar{x}^1, \dots, \bar{u}^k, \bar{x}^k) = \vec{z} \in \bar{\mathcal{X}} \times (\bar{\mathcal{U}} \times \bar{\mathcal{X}})^k = \vec{\mathcal{Z}}$.

Here, we generically refer to each variable in the plan parameter space as z_p where now $p \in \{1, \dots, m + k(m+n)\} = \Theta$. When applying the constraints for clause a_t, plan state $\bar{x}^{t-1}$ is the transition state $\bar{x}$, and likewise $\bar{x}^t$ is $\bar{x}'$. Solutions using this skeleton must satisfy a single clause of all plan-wide constraints $\mathcal{C}_{\vec{a}} = \mathcal{C}_0 \cap \mathcal{C}_{a_1} \cap \cdots \cap \mathcal{C}_{a_k} \cap \mathcal{C}_*$.

Definition 12. A set of constraints $\mathcal{C}$ is *satisfiable* if there exists parameter values $\vec{z} \in \vec{Z}$ such that $\bar{z} \in \bigcap_{C \in \mathcal{C}} C{\uparrow}^{\Theta}$.

Definition 13. A problem $\mathcal{P}$ is *feasible* if there exists a plan skeleton $\vec{a}$ that $\mathcal{C}_{\vec{a}}$ is satisfiable.

For a given plan skeleton, finding satisfying parameter values $\vec{z}$ is a hybrid *CSP*. The joint set of constraints forms a *constraint network*, a bipartite graph between constraints and parameters (Dechter, 1992; Lagriffoul et al., 2014). An edge between a constraint node $C = \langle P, R \rangle$ and state or control node x_p^{t-1}, u_p^t, or x_p^t is defined if and only if $p \in P$. Figure 1 displays a general constraint network. Many transition systems in practice will have constraint networks with many fewer edges because each P contains only a small number of parameters.

4. Example transition systems

We are interested in a general algorithmic framework that can be applied in many factored transition systems. Consider the following two applications and their representation as factored transition systems. We begin with a motion planning application to illustrate the approach, and then describe a pick-and-place application.

4.1. Motion planning

Many motion planning problems may be defined by a bounded configuration space $\mathcal{Q} \subset \mathbb{R}^d$ and collision-free configuration space $\mathcal{Q}_{free} \subseteq \mathcal{Q}$. We will consider planning motions composed of a finite sequence of straight-line trajectories t between waypoints q, q'. Problems are given by an initial configuration $q_0 \in \mathcal{Q}$ and a goal configuration

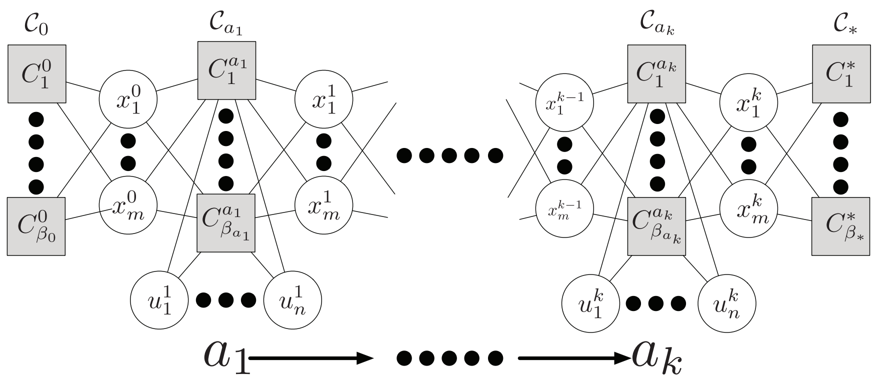


Fig. 1. A constraint network for a generic plan skeleton $\vec{a} = (a_1, \ldots, a_k)$ and parameters $\vec{z} = (\bar{x}^0, \bar{u}^1, \bar{x}^1, \ldots, \bar{u}^k, \bar{x}^k)$.

$q_* \in \mathcal{Q}$. Motion planning can be modeled as a transition system with state space $\bar{\mathcal{X}} = \mathcal{Q}$ and control space $\bar{\mathcal{U}} = \mathcal{Q}^2$. The transition relation $\mathcal{T} = \{\mathcal{C}_{Move}\}$ has a single clause

$$\mathcal{C}_{Move} = \{Motion, CFree\}$$

The transition relation does not exhibit any useful factoring. A motion constraint *Motion* enforces u_t is a straight-line trajectory between x_q and $x_{q'}$:

$$Motion = \langle (x_q, u_t, x'_q), \{(q, t, q') | q, q' \in \mathcal{Q}^2 \\ t(\lambda) = \lambda q + (1 - \lambda) q'\} \rangle$$

A collision-free constraint *CFree* ensures all configurations on the trajectory are not in collision.

$$CFree = \langle (u_t), \{t | \forall \lambda \in [0, 1].\ t(\lambda) \in \mathcal{Q}_{free}\} \rangle$$

The initial clause is $\mathcal{C}_0 = \bar{x}^0 = \{x_q = q_0\}$ and the goal clause is $\mathcal{C}_* = \{x_q = q_*\}$.

The system state could alternatively be described as $\bar{x} = (x_{j_1}, \ldots, x_{j_d})$ where j is a single robot degree of freedom. For simplicity, we combine individual degrees of freedom included within the same constraints into a single variable. This is possible because the set of robot joints always occurs together when mentioned as parameters within motion and kinematic constraints.

Figure 2 displays a constraint network for a plan skeleton of length k. Because the transition relation has a single clause, all solutions have this form. Dark gray circles are parameters, such as the initial and final configurations, that are constrained by constant equality. Free parameters are yellow circles. Constraints are orange rectangles.

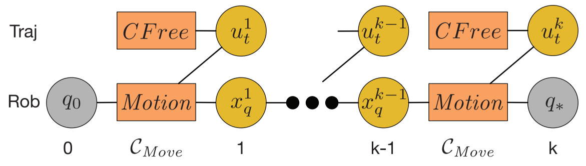


Fig. 2. Motion planning plan skeleton of length k.

performs a grasping operation. As before, the robot can execute straight-line trajectories t between waypoints q, q'.

Pick-and-place problems can be modeled as a transition system with state space $\bar{\mathcal{X}} = \mathcal{Q} \times \text{SE}(3)^{|\mathcal{O}|} \times (\{\textbf{None}\} \cup \mathcal{O})$. States are $\bar{x} = (x_q, x_{o_1}, \ldots, x_{o_{|\mathcal{O}|}}, x_h)$. Let $h \in \mathcal{O}$ indicate that the robot is holding object h and $h = \textbf{None}$ indicate that the robot's gripper is empty. When $h = o$, the pose x_o of object o is given in the end-effector frame. Otherwise, x_o is relative to the world frame. By representing attachment as a change in frame, the pose of an object remains fixed, relative to the gripper, as the robot moves. Controls are pairs $\bar{u} = (u_t, u_g)$ composed of trajectories u_t and Boolean gripper force commands u_g. Let $u_g = \textbf{True}$ correspond to sustaining a grasp force and $u_g = \textbf{False}$ indicate applying no force.

The transition relation $\mathcal{T}$ has $1 + 3|\mathcal{O}|$ clauses because *pick, move-while-holding*, and *place* depend on o:

$$\mathcal{T} = \{\mathcal{C}_{Move}\} \cup \{\mathcal{C}^o_{MoveH}, \mathcal{C}^o_{Pick}, \mathcal{C}^o_{Place} | o \in \mathcal{O}\}$$

Here $\mathcal{C}_{Move}$ and $\mathcal{C}^o_{MoveH}$ clauses correspond to the robot executing a trajectory u_t while its gripper is empty or holding object o:

$$\mathcal{C}_{Move} = \{Motion, CFree, x_h = \textbf{None}, x_h = x_{h'} \\ u_g = \textbf{False}\} \cup \{x_{o'} = x'_{o'}, CFree_{o'} | o' \in \mathcal{O}\} \\ \mathcal{C}^o_{MoveH} = \{Motion, CFreeH_o, x_h = o, x_h = x'_h \\ u_g = \textbf{True}\} \cup \{CFreeH_{o,o'} | o' \in \mathcal{O}, o \neq o'\}$$

where $CFree_o$ is a constraint containing robot trajectories u_t and object o poses x_o that are not in collision with each other, $CFreeH_o$ is a constraint composed of robot

4.2. Pick-and-place planning

A pick-and-place problem is defined by a single robot with configuration space $\mathcal{Q} \subset \mathbb{R}^d$, a finite set of moveable objects $\mathcal{O}$, a set of stable placement poses $\mathcal{S}_o \subset \text{SE}(3)$ for each object $o \in \mathcal{O}$, and a set of grasp poses relative to the end-effector $\mathcal{G}_o \subset \text{SE}(3)$ for each object $o \in \mathcal{O}$. The robot has a single manipulator that is able to rigidly attach itself to a single object at a time when the end-effector g

trajectories u_t and object o grasps x_o relative to the end-effector that are not in collision with the environment, and $CFreeH_{o,o'}$ is a constraint containing robot trajectories u_t, object o grasps x_o relative to the end-effector, and object o' poses $x_{o'}$ that are not in collision with each other.

Here $\mathcal{C}^o_{Pick}$ and $\mathcal{C}^o_{Place}$ clauses correspond to instantaneous changes in what the gripper is holding:

$$\mathcal{C}^o_{Pick} = \{Stable_o, Grasp'_o, Kin_o, x_q = x'_q, x_h = \textbf{None}, x_{h'} = o\} \cup \{x_{o'} = x'_{o'} | o' \in \mathcal{O}, o \neq o'\}$$

$$\mathcal{C}^o_{Place} = \{Grasp_o, Stable'_o, Kin'_o, x_q = x'_q, x_h = o, x'_h = \textbf{None}\} \cup \{x_{o'} = x'_{o'} | o' \in \mathcal{O}, o \neq o'\}$$

As a result, they do not involve any control variables. Here $Grasp_o = \langle (x_o), \mathcal{G}_o \rangle$ is a constraint that x_o is a grasp transform from the object frame to the end-effector frame. Let $Grasp'_o = \langle (x'_o), \mathcal{G}_o \rangle$ be the same constraint but on x'_o. Similarly, $Stable_o = \langle (x_o), \mathcal{S}_o \rangle$ is a constraint that x_o is a stable placement, and $Stable'_o = \langle (x'_o), \mathcal{S}_o \rangle$. Finally, Kin_o is a constraint composed of kinematic solutions involving object o for a grasp g, pose p, and robot configuration q:

$$Kin_o = \langle (x'_o, x_o, x_q), \{(g, p, q) | \textsc{kin}(q) = pg^{-1}\} \rangle$$

Here Kin'_o is the equivalent constraint but on state variables (x_o, x'_o, x_q), where x_o and x'_o are swapped. Because Kin_o and Kin'_o refer to the same relation and only involve different parameters, we will just refer to Kin_o.

Pick-and-place transition systems are substantially factorable. Each constraint involves at most three variables. In addition, in each clause, many variables are entirely constrained by equality. For $\mathcal{C}_{Move}, \mathcal{C}^o_{MoveH}$ clauses, only half of the variables are not constrained by equality: $\{x_q, u_t, x_{q'}\} \cup \{x_{o'} | o' \in \mathcal{O}\}$. For $\mathcal{C}^o_{Pick}, \mathcal{C}^o_{Place}$ clauses, only $\{x_o, x'_o, x_q\}$ variables are not constrained by equality.

We use the pick-and-place problem shown in Figure 3 with two movable objects A, B as a running example. The initial state $\bar{x}^0 = (q_0, a_0, b_0, \textbf{None})$ is fully specified using equality constraints $\mathcal{C}_0 = \{\bar{x}_q = q_0, \bar{x}_A = a_0, \bar{x}_B = b_0, \bar{x}_h = \textbf{None}\}$. We assume that a_0 and b_0 are stable poses: $a_0 \in \mathcal{S}_A$ and $b_0 \in \mathcal{S}_B$. The goal states $\bar{X}^*$ are given as constraints $\mathcal{C}_* = \{Region, x_q = q_*\}$ where the region constraint $Region_A = \langle (x_A), \mathcal{R}_A \rangle$, for $\mathcal{R}_A \subseteq \mathcal{S}_A$ is a subset of the stable placements for object A.

A useful consequence of factoring is that the same control values $\bar{u}$ can be considered in many transitions. Consider the two candidate transitions in figure 4, both using the same control trajectory u_t. The application of u_t in the left figure results in a valid transition for clause *Move*. In fact, for a majority of the combinations of placements of x_A, x_B, u_t is a valid transition. Thus, for a single value of u_t, we are implicitly representing many possible transitions. The right figure, however, shows an instance in which this u_t does not correspond to a legal transition as it would result in a collision with B, thus violating $CFree_B$.

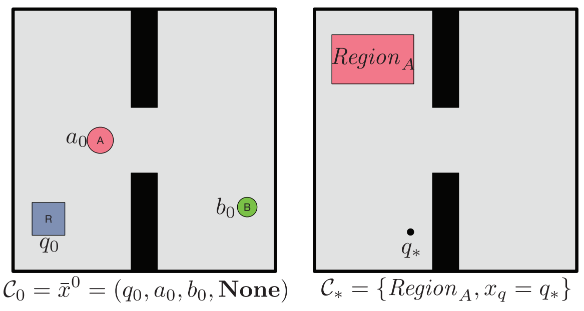

a_0
A
R
q_0
b_0
B
$\mathcal{C}_0 = \bar{x}^0 = (q_0, a_0, b_0, \textbf{None})$
$Region_A$
q_*
$\mathcal{C}_* = \{Region_A, x_q = q_*\}$

Fig. 3. The initial state (left) and goal constraints (right) for a pick-and-place problem involving a square robot R and movable circle objects A and B.

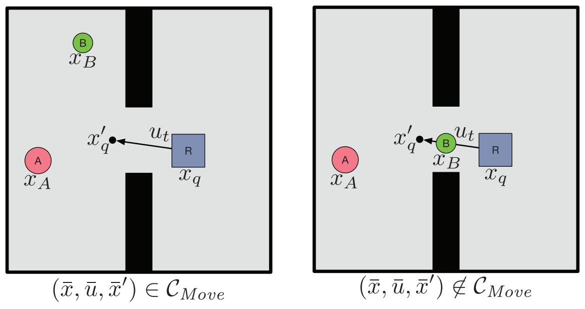

B
x_B
x'_q
u_t
R
x_q
A
x_A
$(\bar{x}, \bar{u}, \bar{x}') \in \mathcal{C}_{Move}$
$(\bar{x}, \bar{u}, \bar{x}') \notin \mathcal{C}_{Move}$

Fig. 4. A valid transition (left) and an invalid transition (right) for the same control trajectory u_t. The right transition is invalid because it violates the $CFree_B$ collision-free constraint.

Figure 5 displays the constraint network for a plan skeleton $\vec{a} = (Move, Pick^A, MoveH^A, Place^A, Move)$ that grabs A, places A in $Region_A$, and moves the robot to q_*. Thick edges indicate pairwise equality constraints. Light gray parameters are transitively fixed by pairwise equality. We will omit the constraint subscripts for simplicity. Despite having 29 total parameters, only 7 are free parameters. This highlights the strong impact of equality constraints on the dimensionality of the plan parameter space.

5. Sampling-based planning

Constraints involving continuous variables are generally characterized as uncountably infinite sets, which are often difficult to reason with explicitly. Instead, each constraint can be described using a blackbox, implicit *test*. A test for constraint $C = \langle P, R \rangle$ is a Boolean-valued function $t_C : \mathcal{Z}_P \to \{0, 1\}$ where $t_C(\bar{z}_P) = [\bar{z}_P \in R]$. Implicit representations are used in sampling-based motion planning, where they replace explicit representations of complicated robot and environment geometries with collision-checking procedures.

To use tests, we need to produce potentially satisfying values for $\bar{z}_P = (z_{p_1}, \ldots, z_{p_k})$ by sampling $\mathcal{Z}_{p_1}, \ldots, \mathcal{Z}_{p_k}$. Thus, we still require an explicit representation for $\mathcal{X}_1, \ldots, \mathcal{X}_m$ and $\mathcal{U}_1, \ldots, \mathcal{U}_n$; however, these are typically

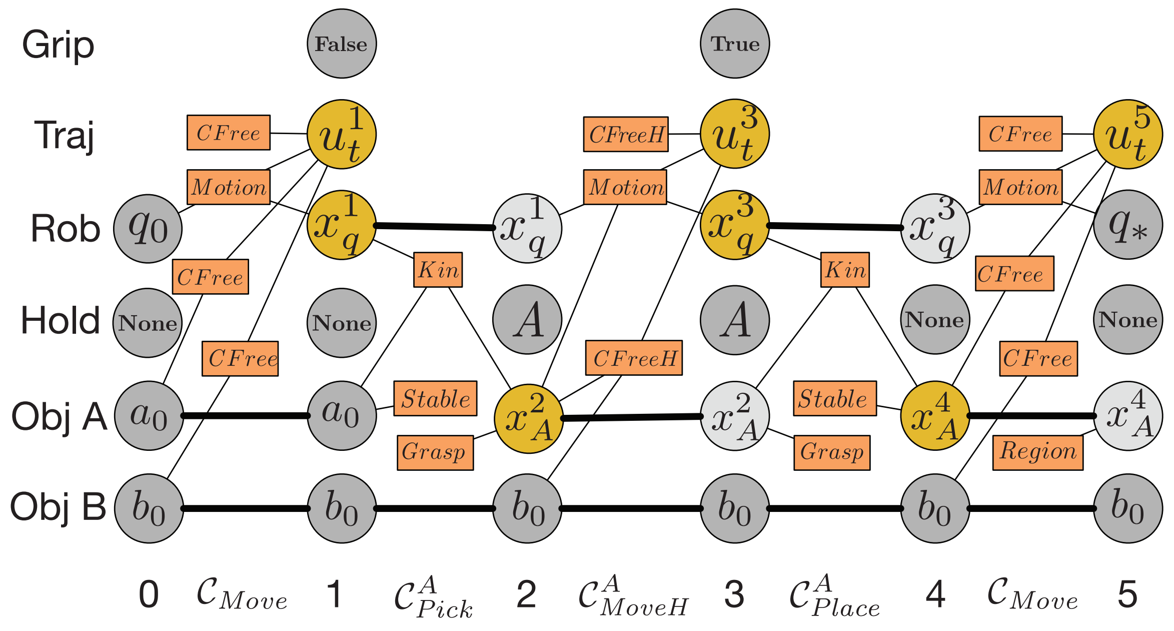


Fig. 5. Pick-and-place constraint network for a plan skeleton $\vec{a} = (Move, Pick^A, MoveH^A, Place^A, Move)$.

less difficult to characterize. We assume $\mathcal{X}_1, \ldots, \mathcal{X}_m$, $\mathcal{U}_1, \ldots, \mathcal{U}_n$ are each bounded manifolds. This strategy of sampling variable domains and testing constraints is the basis of *sampling-based planning* (Kavraki et al., 1996). These methods draw values from $\mathcal{X}_1, \ldots, \mathcal{X}_m$ and $\mathcal{U}_1, \ldots, \mathcal{U}_n$ using deterministic or random *samplers* for each space and test which combinations of sampled values satisfy required constraints.

Sampling-based techniques are usually not complete over all problem instances. First, they cannot generally identify and terminate on infeasible instances. Second, they are often unable to find solutions to instances that require identifying values from a set that has zero measure in the space from which samples are being drawn. In motion planning, these are problems in which all paths have zero *clearance* (sometimes called *path-goodness*), the infimum over distances from the path to obstacles (Kavraki et al., 1998; Kavraki and Latombe, 1998). Thus, sampling-based motion planning algorithms are only theoretically analyzed over the set of problems admitting a path with positive clearance. Conditions similar to positive clearance include an open configuration space (Laumond, 1998), positive *ε-goodness* (Barraquand et al., 1997; Kavraki and Latombe, 1998; Kavraki et al., 1995), and expansiveness (Hsu et al., 1997; Kavraki and Latombe, 1998).

The ideas of positive clearance (Garrett et al., 2017a; Van Den Berg et al., 2009), positive ε-goodness (Barry, 2013; Berenson and Srinivasa, 2010), and expansiveness (Hauser and Latombe, 2010; Hauser and Ng-Thow-Hing, 2011) have been extended to several manipulation planning contexts. In manipulation planning, zero-clearance related properties can take on additional forms. For instance, a goal constraint that two movable objects are placed within a tight region may only admit pairs of goal poses lying on a submanifold of the full set of stable placements (Garrett et al., 2017a; Van Den Berg et al., 2009). At the same time, kinematic constraints resulting from a pick operation define a set lying on a submanifold of the robot's configuration space (Berenson and Srinivasa, 2010; Garrett et al., 2017a). Thus, these clearance- related properties typically identify a set of anticipated submanifolds, such as a set of inverse kinematic solutions, that can be sampled directly.

More generally, sampling-based algorithms are typically only complete over *robustly feasible* problems (Karaman and Frazzoli, 2011). When directly applied to factored transitions, a problem is robustly feasible if there exists a plan skeleton $\vec{a}$ such that $\bigcap_{C \in \mathcal{C}_{\vec{a}}} C{\uparrow}^{\Theta}$ contains an open set in plan-parameter-space $\bar{\mathcal{Z}}$.

5.1. Dimensionality-reducing constraints

Some problems of interest involve individual constraints that only admit a set of values on a lower-dimensional subset of their parameter spaces. A *dimensionality-reducing constraint* $C = \langle P, R \rangle$ is one in which $R \subseteq \bar{\mathcal{Z}}_P$ does not contain an open set. Consider the $Stable_o$ constraint. The set of satisfying values lies on a three-dimensional manifold. By our current definition, all plans involving this constraint are not robustly feasible. When a problem involves dimensionality-reducing constraints, we have no choice but to sample at their intersection. This, in general, requires an explicit characterization of their intersection, which we may not have. Moreover, the number of dimensionality-reducing constraint combinations can be unbounded as plan skeletons may be arbitrarily long. However, in some cases, we can produce this intersection automatically using explicit characterizations for only a few spaces.

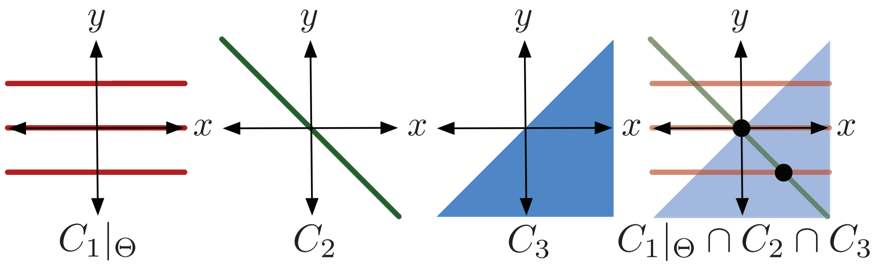


Fig. 6. Intersection of three constraints, two of them dimensionality-reducing.

We motivate these ideas with an example (Figure 6). Consider a plan skeleton $\vec{a}$ with parameters $\Theta = (z_x, z_y)$ where $\mathcal{Z}_x = \mathcal{Z}_y = (-2, +2)$ and constraints $\mathcal{C}_{\vec{a}} = \{C_1, C_2, C_3\}$ where

$$C_1 = \langle (y), \{-1, 0, 1\} \rangle$$
$$C_2 = \langle (x, y), \{(x, y) | x + y = 0\} \rangle$$
$$C_3 = \langle (x, y), \{(x, y) | x - y \geqslant 0\} \rangle$$

The set of solutions $C_1 \uparrow^\Theta \cap C_2 \cap C_3 = \{(1, -1), (0, 0)\}$ is zero-dimensional whereas the parameter space is two-dimensional. This is because C_1 and C_2 are both dimensionality-reducing constraints. A uniform sampling strategy where $X, Y \sim \text{Uniform}(-2, +2)$ has zero probability of producing a solution.

To solve this problem using a sampling-based approach, we must sample from $C_1 \uparrow^\Theta \cap C_2$. Suppose we are unable to analytically compute $C_1 \uparrow^\Theta \cap C_2$, but we do have explicit representations of C_1 and C_2 independently. In particular, suppose we know C_2 conditioned on values of y, $C_2(y) = \langle (x), \{-y\} \rangle$. Now, we can characterize $C_1 \uparrow^\Theta \cap C_2 = \{(x, y) | y \in R_1, x \in R_2(y)\} = \{(1, -1), (0, 0), (-1, 1)\}$. With respect to a counting measure on this discrete space, $C_1 \uparrow^\Theta \cap C_2 \cap C_3$ has positive measure. This not only gives a representation for the intersection but also suggests the following way to sample the intersection: $Y \sim \text{Uniform}(\{-1, 0, +1\})$, $X = -Y$, and reject (X, Y) that does not satisfy C_3. This strategy is not effective for all combinations of dimensionality-reducing constraints. Suppose that instead $C_1 = \langle (x, y), \{(x, y) | x - y = 1\} \rangle$. Because both constraints involve x and y, we are unable to condition on the value of one parameter to sample the other.

5.2. Intersection of manifolds

In this section, we develop the topological tools to generalize the previous example. Our objective is to show that by making assumptions on each dimensionality-reducing constraint individually, we can understand the space formed by the intersection of many dimensionality constraints. First, we overview several topological ideas that we will use (Tu, 2010).

5.2.1. Topological tools. A d-dimensional *manifold* M is a topological space that is locally homeomorphic to d-dimensional Euclidean coordinate space. Let $d = \dim M$ be the dimension of the coordinate space of M. An *atlas* for an d-dimensional manifold $M \subseteq \mathbb{R}^m$ is a set of charts $\{(U_\alpha, \varphi_\alpha), \ldots\}$ such that $\bigcup_\alpha U_\alpha = M$. Each *chart* $(U_\alpha, \varphi_\alpha)$ is given by an open set $U_\alpha \subseteq M$ and a *homeomorphism*, a continuous bijection with a continuous inverse, $\varphi : U_\alpha \to \mathbb{R}^d$. Let N be a regular submanifold of ambient manifold M. Then, $\text{codim}\, N = \dim M - \dim N$ is the *codimension* of N.

Define $T_x(M)$ to be the *tangent space* of manifold M at $x \in M$, which intuitively contains the set of directions in which one can pass through M at x. A smooth map of manifolds $f : M \to N$ is *submersion* at $x \in M$ if its *differential*, a linear map between tangent spaces, $df_x : T_x(M) \to T_{f(x)}(N)$ is surjective. When $\dim M \geqslant \dim N$, this is equivalent to the Jacobian matrix of f at x having maximal rank equal to $\dim N$. When f is a submersion, by the *preimage theorem* (also called the implicit function theorem and the regular level set theorem), the preimage $f^{-1}(y)$ for any $y \subseteq N$ is a submanifold of M of dimension $\dim M - \dim N$. Similarly, by the *local normal submersion theorem*, there exists coordinate charts φ and φ' local to $x \in M$ and $f(x) \in N$ such that f is a projection in coordinate space. Thus, the first $\dim N$ coordinates of $\varphi(x)$ and $\varphi(f(x))$ are the same and $\varphi(x)$ contains an extra codim N coordinates.

For manifolds N_1 and N_2, consider projections $\text{proj}_1 : (N_1 \times N_2) \to N_1$ and $\text{proj}_2 : (N_1 \times N_2) \to N_2$. For all $\bar{y} = (y_1, y_2) \in N_1 \times N_2$, the map

$$(d\,\text{proj}_1, d\,\text{proj}_2) : T_{\bar{y}}(N_1 \times N_2) \to T_{y_1}(N_1) \times T_{y_2}(N_2)$$

is an isomorphism. Thus, the tangent space of a product manifold is isomorphic to the Cartesian product of the tangent spaces of its components.

Let N_1, N_2 both be submanifolds of the same ambient manifold M. An intersection is *transverse* if $\forall x \in (N_1 \cap N_2).\ T_x(N_1) + T_x(N_2) = T_x(M)$ where

$$T_x(N_1) + T_x(N_2) = \{v + w | v \in T_x(N_1), w \in T_x(N_2)\}$$

If the intersection $N_1 \cap N_2$ is transverse, then $N_1 \cap N_2$ is a submanifold of codimension

$$\text{codim}(N_1 \cap N_2) = \text{codim } N_1 + \text{codim } N_2$$

Intuitively, the intersection is transverse if their combined tangent spaces produce the tangent space of the ambient manifold.

5.2.2. Conditional constraints. We start by defining conditional constraints, a binary partition of P for a constraint $\langle P, R\rangle$.

Definition 14. A *conditional constraint* $\langle I, O, R\rangle$ for a constraint $C = \langle P, R\rangle$ is a partition of P into a set of *input parameters* I and a set of *output parameters* O.

Define $\text{proj}_I(\bar{z}) = \bar{z}_P$ to be the set-theoretic projection of $\bar{z}$ onto parameters P. Let its inverse, the projection preimage $\text{proj}_I^{-1}(\bar{z}_I)$, be the following:

$$\text{proj}_I^{-1}(\bar{z}_I) = \{\bar{z}_P \in R | \bar{z}_P = (\bar{z}_I, \bar{z}_O)\}$$

Conditional constraints will allow us to implicitly reason about intersections of $\text{proj}_I(R)$ rather than R directly. To cleanly describe a lower-dimensional space produced by the intersection of several constraints, we will consider a simplified manifold M that is a subset of R. In addition, we will make an assumption regarding the relationship between $\text{proj}_I(M)$ and M to use the preimage theorem.

Definition 15. A *conditional constraint manifold* $\langle I, O, M\rangle$ for a conditional constraint $\langle I, O, R\rangle$ is a nonempty smooth manifold $M \subseteq R$ such that $\text{proj}_I : M \to \mathcal{Z}_I$ is a submersion $\forall \bar{z}_P \in M$.

In our context, M is a submanifold of $\bar{\mathcal{Z}}_P = \mathcal{Z}_{P_1} \times \cdots \times \mathcal{Z}_{P_{|P|}}$ because $R \subset \bar{\mathcal{Z}}_P$. The projection $\text{proj}_I = f$ is trivially smooth map between manifold M and product manifold $\bar{\mathcal{Z}}_I$. Thus, proj_I is a submersion at $\bar{z}_P \in M$ when $d\text{proj}_I : T_{\bar{z}_P}(M) \to T_{\bar{z}_I}(\mathcal{Z}_I)$ is surjective. When $\mathcal{Z}_I = \mathbb{R}^d$, this is equivalent to the subspace of tangent space $T_{\bar{z}_P}(M)$ on parameters I having rank d. A more intuitive interpretation is locally at $\bar{z}_P$, there exists coordinates to control M in any direction of I. This restriction is used to ensure that the intersection of M and other conditional constraint manifolds will not be transdimensional. This implies the preimage $\text{proj}_I^{-1}(\bar{z}_I)$ for particular values of input parameters $\bar{z}_I \in \text{proj}_I(M)$ is a submanifold of M of codimension $\dim \bar{\mathcal{Z}}_I$. Importantly, $\text{proj}_I(M)$ is an open set within $\bar{\mathcal{Z}}_I$. This is the key consequence of the submersion assumption, which is useful when intersecting arbitrary conditional constraint manifolds.

Suppose we are given a set of conditional constraint manifolds $\{\langle I_1, O_1, M_1\rangle, \ldots, \langle I_n, O_n, M_n\rangle\}$. Let $\Theta = \bigcup_{j=1}^n P_j$ be the set of parameters they collectively mention. Let $S = \bigcap_{j=1}^n M_j \uparrow^{\Theta}$ be their intersection when each constraint is extended on Θ. We now present the main theorem, which gives a sufficient condition for when S is a submanifold of $\bar{\mathcal{Z}}_\Theta$. This theorem is useful because it identifies when the intersection of several possibly dimensionality-reducing constraints is a space that we can easily characterize.

Theorem 1. *If $\{O_1, \ldots, O_n\}$ is a partition of Θ and there exists an ordering of $(\langle I_1, O_1, M_1\rangle, \ldots, \langle I_n, O_n, M_n\rangle)$ such that $\forall i = \{1, \ldots, n\}$ $I_i \subseteq \bigcup_{j=1}^{i-1} O_j$, then $S = \bigcap_{j=1}^n M_j \uparrow^{\Theta}$ is a submanifold of $\bar{\mathcal{Z}}_\Theta$ of codimension $\sum_{j=1}^n$ codim M_j.*

Proof. Define $\Theta_i = \bigcup_{j=1}^i P_j$ to be union of the first i sets of parameters. Note that $\Theta_i = \bigcup_{j=1}^i O_j$ follows from the second assumption. Let $S_i = \bigcap_{j=1}^i M_j \uparrow^{\Theta_i}$ be the intersection of the first i constraint manifolds over parameters Θ_i. On the final intersection, $\Theta_n = \Theta$ and $S_n = S$. We can also write S_i recursively as $S_i = S_{i-1} \uparrow^{\Theta_i} \cap M_i \uparrow^{\Theta_i}$ where $S_0 = \emptyset$. Here, S_{i-1} and M_i are extended by Cartesian products with $\bar{\mathcal{Z}}_{O_i}$ and $\bar{\mathcal{Z}}_{\Theta_i \setminus P_i}$, respectively.

We proceed by induction on i. For the base case, $I_1 = \bigcup_{j=1}^0 O_j = \emptyset$. Thus, $S_0 \uparrow^{\Theta_1} = \bar{\mathcal{Z}}_{O_1}$ and $M_1 \uparrow^{\Theta_1} = M_1$. As $M_1 \subseteq \bar{\mathcal{Z}}_{O_1}$,

$$S_1 = (S_0 \uparrow^{\Theta_1} \cap M_1 \uparrow^{\Theta_1}) = (\bar{\mathcal{Z}}_{O_1} \cap M_1) = M_1$$

is a submanifold of codimension codim M_1 within $\bar{\mathcal{Z}}_{O_1}$.

For the inductive step, we assume that after the $(i - 1)$th intersection, S_{i-1} is a submanifold of codimensionality codim S_{i-1} on parameters Θ_{i-1}. We will show that S_i is a manifold of codimension codim $S_{i-1} + \dim M_i$ on parameters Θ_i. By isomorphism of product manifold tangent spaces, $\forall \bar{z} \in S_{i-1} \uparrow^{\Theta_i}$, $T_{\bar{z}}(S_{i-1} \uparrow^{\Theta_i})$ is isomorphic to $T_{\bar{z}_{\Theta_{i-1}}}(S_{i-1}) \times T_{\bar{z}_{O_i}}(\bar{\mathcal{Z}}_{O_i})$. Similarly, $\forall \bar{z} \in M_i \uparrow^{\Theta_i}$, $T_{\bar{z}}(M_i \uparrow^{\Theta_i})$ is isomorphic to $T_{\bar{z}_{P_i}}(M_i) \times T_{\bar{z}_{\Theta_i \setminus P_i}}(\bar{\mathcal{Z}}_{\Theta_i \setminus P_i})$. As a result, the projection for the extended constraint $d\text{proj}_{\Theta_{i-1}} : T_{\bar{z}}(M_i \uparrow^{\Theta_i}) \to T_{\bar{z}_{\Theta_{i-1}}}(\bar{\mathcal{Z}}_{\Theta_{i-1}})$ is itself surjective for all $\bar{z}$ and therefore is a submersion.

Because $S_i \subseteq \bar{\mathcal{Z}}_{\Theta_i}$, for any $\bar{z} \in S_i$, $T_{\bar{z}}(M_i \uparrow^{\Theta_i}) + T_{\bar{z}}(S_{i-1} \uparrow^{\Theta_i}) \subseteq T_{\bar{z}}(\bar{\mathcal{Z}}_{\Theta_i})$ For each $\bar{z} \in S_i$, consider any tangent $v \in T_{\bar{z}}(\bar{\mathcal{Z}}_{\Theta_i})$. Let $v = x + y$ be an orthogonal decomposition of v into a component x defined on parameters Θ_{i-1} and component y defined on O_i. The first component x is isomorphic to an element of $T_{\bar{z}}(M_i \uparrow^{\Theta_i})$ because its tangent space is surjective to $\bar{\mathcal{Z}}_{\Theta_{i-1}}$. The second component y is isomorphic to an element of $T_{\bar{z}}(S_{i-1} \uparrow^{\Theta_i})$ by the isomorphism to the product space involving $\bar{\mathcal{Z}}_{O_i}$. Thus, $T_{\bar{z}}(\bar{\mathcal{Z}}_{\Theta_i}) \subseteq T_{\bar{z}}(M_i \uparrow^{\Theta_i}) + T_{\bar{z}}(S_{i-1} \uparrow^{\Theta_i})$. As a result, $M_i \uparrow^{\Theta_i}$ and $S_{i-1} \uparrow^{\Theta_i}$ intersect transversally implying that S_i is a smooth submanifold of $\bar{\mathcal{Z}}_{\Theta_i}$ with codimension

$$\text{codim } M_i \uparrow^{\Theta_i} + \text{codim } S_{i-1} \uparrow^{\Theta_i} = \text{codim } M_i + \text{codim } S_{i-1}$$

After n iterations, the entire intersection is a submanifold of codimension codim S = codim $M_1 + \cdots +$ codim M_n in Θ. Let $d_i = \dim M_i - \dim \bar{\mathcal{Z}}_{I_i}$ be the dimension of the submanifold defined by the projection preimage $\text{proj}_{I_i}^{-1}(\bar{z}_I)$. Finally, $\dim S$ can be seen alternatively as the sum of

$\sum_{i=1}^{n} d_i$, the number of coordinates introduced on each iteration;

$$\begin{aligned}\dim S &= \dim \bar{\mathcal{Z}}_\Theta - \sum_{i=1}^{n} \text{codim } M_i \\ &= \dim \bar{\mathcal{Z}}_\Theta - \sum_{i=1}^{n} (\dim \bar{\mathcal{Z}}_{P_i} - \dim M_i) \\ &= \dim \bar{\mathcal{Z}}_\Theta - \sum_{i=1}^{n} (\dim \bar{\mathcal{Z}}_{O_i} + \dim \bar{\mathcal{Z}}_{I_i} - \dim M_i) \\ &= \sum_{i=1}^{n} (\dim M_i - \dim \bar{\mathcal{Z}}_{I_i}) = \sum_{i=1}^{n} d_i\end{aligned}$$

We will call S a *sample space* when Theorem 1 holds. From the partition condition, each parameter must be the output of exactly one conditional constraint manifold. Intuitively, a parameter can only be "chosen" once. From subset condition, each input parameter must be an output parameter for some conditional constraint manifold earlier in the sequence. Intuitively, a parameter must be "chosen" before it can be used to produce values for other parameters. Theorem 1 can be understood graphically using *sampling networks*. A sampling network is a subgraph of a constraint network using constraints corresponding to conditional constraint manifolds. The graphical relationship between a constraint network and a sampling network is analogous to the relationship between a factor graph and a Bayesian network from probabilistic inference (Jensen, 1996). However, this resemblance is purely structural because constraint networks and sampling networks represent sets given as the intersection of several constraints while graphical models represent joint distributions over sets. Each parameter node in a sampling network has exactly one incoming edge. Directed edges go from input parameters to constraints or constraints to output parameters. Each parameter is the output of exactly one constraint. Finally, the graph is acyclic.

When analyzing robustness properties, we will assume that we are given a set of conditional constraint manifolds $\mathcal{M}$. This set $\mathcal{M}$ is typically composed of dimensionality-reducing constraints that have a known analytic form. We may have multiple conditional constraint manifolds per constraint C, resulting from the different ways of conditionalizing C. Implicit variable domain constraints $\langle (), (x_i), \mathcal{X}_i \rangle, \langle (), (u_j), \mathcal{U}_i \rangle \in \mathcal{M}$ for each variable are always present within $\mathcal{M}$. Given a set of constraints $\mathcal{C}$ defined on parameters Θ, we can produce the corresponding set of conditional constraint manifolds on Θ by substituting constraints for the conditional constraint manifolds. For a constraint $C = \langle P, R \rangle$, let $\mathcal{M}_C$ be the set of conditional constraint manifolds $\{\langle I, O, M \rangle, \ldots\} \subseteq \mathcal{M}$ associated with C by substituting the input and output parameters I, O for each conditional constraint manifold for with the parameters for P. For a set of constraints, let $\mathcal{M}_{\mathcal{C}} = \cup_{C \in \mathcal{C}} \mathcal{M}_C$ be the union for each constraint $C \in \mathcal{C}$.

Now we generalize our definition of robust feasibility to be with respect to a set of conditional constraint manifolds $\mathcal{M}$. Note that when $\mathcal{M}$ is solely composed of variable domain constraints, the new definition is equivalent to the previous definition. Specifying $\mathcal{M}$ allows us to analyze the set of solutions in a lower-dimensional space S given by the constraint manifolds. Intuitively, a set of constraints is robustly satisfiable $\mathcal{C}$ if for some parameter values $\bar{z} \in S$, all parameter values $\bar{z}'$ in a neighborhood of $\bar{z}$ satisfy $\mathcal{C}$.

Definition 16. A set of constraints $\mathcal{C}$ is robustly *satisfiable* with respect to conditional constraint manifolds $\mathcal{M}$ if there exists a sample space $S = \bigcap_{i=1}^{n} \widehat{M}_i$ formed from conditional constraint manifolds $\{\langle I_1, O_1, M_1 \rangle, \ldots, \langle I_n, O_n, M_n \rangle\} \subseteq \mathcal{C}_{\mathcal{M}}$ where there exists a satisfying $\vec{z}$ with a neighborhood of parameter values $\mathcal{N}(\vec{z})$ in S such that $\mathcal{N}(\bar{z}) \subseteq \bigcap_{C \in \mathcal{C}} C{\uparrow}^{\Theta}$.

We are typically interested in evaluating robustness properties with respect to $\mathcal{M}$ for a set of problems defined with the same transition system $\mathcal{S}$ instead of for just a single problem $\mathcal{P}$. Thus, we a define a domain to be a set of problems $\mathcal{D}$ to be analyzed with respect to a common set of conditional constraint manifolds $\mathcal{M}$. A domain here is similar to the notion of a domain in automated planning (McDermott et al., 1998) because both fix the dynamics of the system and the objective criteria across a set of problems.

Definition 17. A *domain* $\langle \mathcal{D}, \mathcal{M} \rangle$ is given by a set of problems $\mathcal{D}$, where each $\mathcal{P} \in \mathcal{D}$ has an identical transition system $\mathcal{S}$, and a set of conditional constraint manifolds $\mathcal{M}$.

This allows us to describe the robustly feasible subset of problems within a domain.

Definition 18. A factored transition problem $\mathcal{P} \in \mathcal{D}$ within a domain $\langle \mathcal{D}, \mathcal{M} \rangle$ is *robustly feasible* if there exists a plan skeleton $\vec{a}$ such that $\mathcal{C}_{\vec{a}}$ is robustly satisfiable with respect to $\mathcal{M}$.

5.3. Robust motion planning

Our motion planning system involves a single dimensionality reducing constraint *Motion*. The implicit variable domain constraint $Var_q = \langle (x_q), \mathcal{Q} \rangle$ has full dimensionality by default. Each straight-line trajectory t is uniquely described by its start configuration $q = t(0)$ and end configuration $q' = t(1)$. Thus, we can notate a straight-line trajectory as $t = (q, q') \in \mathcal{Q}^2 \subseteq \mathbb{R}^{2d}$. As a result, *Motion* is a $2d$-dimensional submanifold of $\mathcal{Q}^4 \subseteq \mathbb{R}^{4d}$. We consider sample spaces resulting from constraint manifolds $\mathcal{M} = \{Var_q, Motion\}$.

Figure 7 shows a sampling network for the generic motion planning constraint network in Figure 2. This sampling network uses conditional constraint manifolds $\{\langle (), (x_q), Var_q \rangle, \langle (x_q, x_q'), (u_t), Motion \rangle\}$. The projection $\text{proj}_{x_q, x_q'}(Motion) = \mathcal{Q}^2$ and is thus trivially a submersion. The projection preimage $\text{proj}_{x_q, x_q'}^{-1}(q, q') = \{t\}$ is a single point corresponding to the straight-line trajectory. Thus, the sample space $S \subseteq \mathcal{Q}^{3k-1} \subseteq \mathbb{R}^{d(3k-1)}$ is a submanifold of

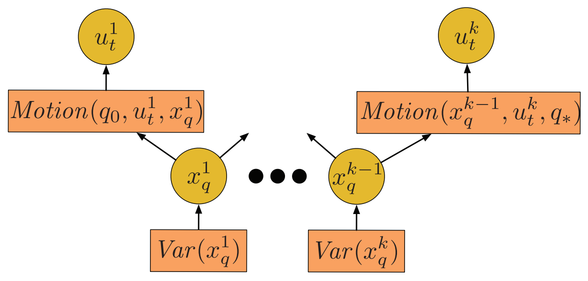


Fig. 7. Motion planning sampling network for the constraint network in Figure 2.

dimensionality $d(k-1)$. Intuitively, the space of satisfying values is parametrizable by each configuration x_q^i for $i = 1, \dots, (k-1)$. As a result, a possible coordinate space of S is $\mathcal{Q}^{k-1}$ corresponding to Cartesian product of $\mathcal{Q}$, the variable domain for each x_q^i.

Subject to this sample space, we can analyze robustly feasible motion planning problems. Figure 8 fixes the plan skeleton $\vec{a} = (Move, Move)$ and investigates robustness properties of four problems varying the environment geometry. The plan skeleton has the following free parameters: $\{u_t^1, x_q^1, u_t^2\}$. However, u_t^1, u_t^2 can be uniquely determined given x_q^1. The choices of these parameters must satisfy the following constraints:

$$\{Motion(q_0, u_t^1, x_q^1), CFree(u_t^1), Motion(x_q^1, u_t^2, q_*), CFree(u_t^2)\}$$

Varying the environment only affects the CFree constraint. The top row displays a top-down image of the scene and the bottom image shows the robot's collision-free configuration space $\mathcal{Q}_{free}$ in light gray. Linear trajectories contained within the light gray regions satisfy their CFree constraint. The yellow region indicates values of x_q^1 that will result in a plan. Problem 1 is unsatisfiable because no values of x_q^1 result in a plan. Problem 2 has only a one-dimensional interval of plan, thus it is satisfiable but not robustly satisfiable. Problem 3 has a two-dimensional region of solutions, so it is robustly satisfiable. Problem 4 is unsatisfiable for the current plan skeleton. However, it is robustly satisfiable for a plan skeleton $\vec{a} = (Move, Move, Move)$.

5.4. Robust pick-and-place

In pick-and-place problems, Stable, Region, Grasp, Kin, and Motion are all individually dimensionality-reducing constraints. Fortunately, we generally understand explicit representations of these sets. We will consider the following constraint manifolds:

$$\mathcal{M} = \{Var_q, Motion\} \cup \{Stable_o, Grasp_o, Kin_o | o \in \mathcal{O}\}$$

We will only consider problems in which $\dim Stable_o = \dim Region_o$ in which case $Stable_o$ captures the reduction of dimensionality from $Region_o$. Again, $CFree$, $CFree_o$, $CFreeH_o$, and $CFreeH_{o,o'}$ are not assumed to be dimensionality-reducing constraints.

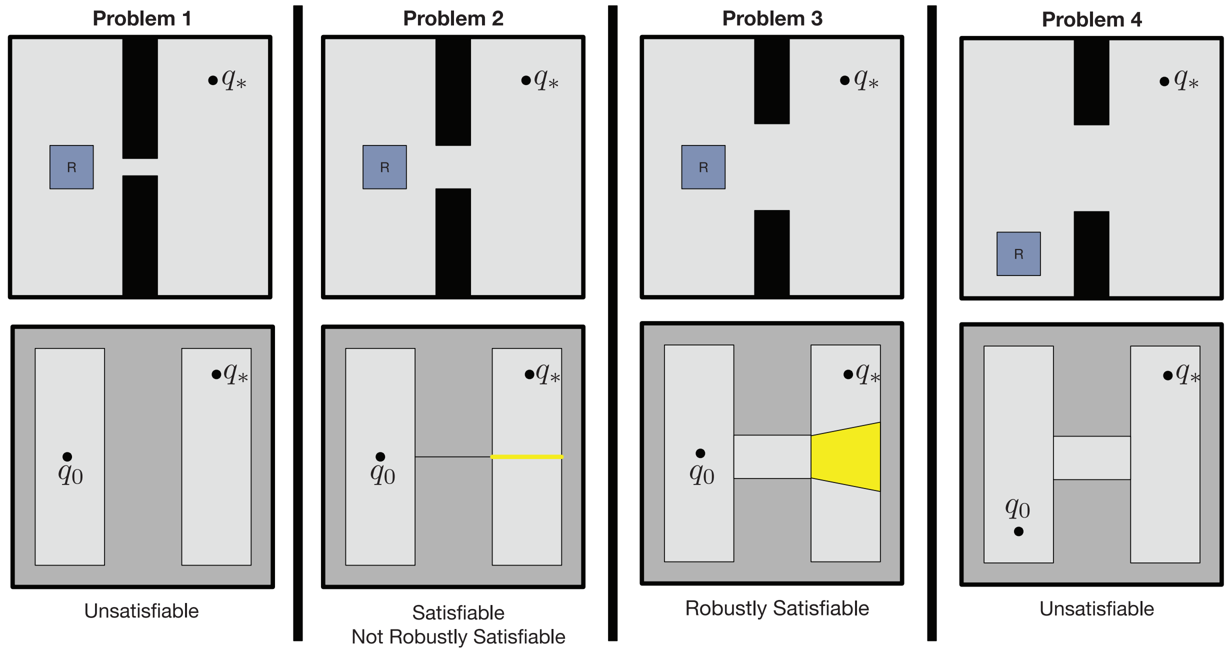
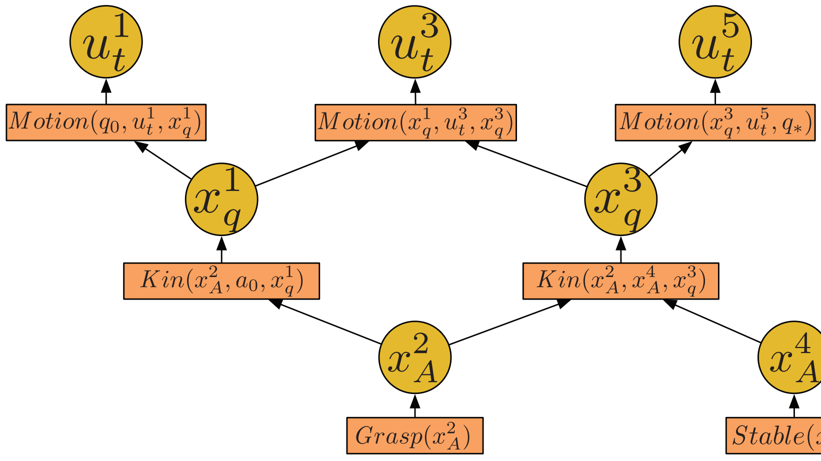


Fig. 8. From left to right, an unsatisfiable, satisfiable (but not robustly satisfiable), robustly satisfiable, and unsatisfiable motion planning problem for a plan skeleton involving two transitions. The top row displays each problem, varying the environment. The bottom row displays the collision-free configuration space $\mathcal{Q}_{free}$. Yellow regions of configuration space correspond to values of x_q^1 satisfying the plan skeleton.



u_t^1
u_t^3
u_t^5
$Motion(q_0, u_t^1, x_q^1)$
$Motion(x_q^1, u_t^3, x_q^3)$
$Motion(x_q^3, u_t^5, q_*)$
x_q^1
x_q^3
$Kin(x_A^2, a_0, x_q^1)$
$Kin(x_A^2, x_A^4, x_q^3)$
x_A^2
x_A^4
$Grasp(x_A^2)$
$Stable(x_A^4)$

Fig. 9. Pick-and-place sampling network for the constraint network in Figure 5.

Figure 9 shows a sampling network for the pick-and-place constraint network in Figure 5. It uses the following conditional constraint manifolds:

$$\{\langle(x_q, x_q'), (u_t), Motion\rangle\} \cup \{\langle(), (x_o), Stable_o\rangle,$$
$$\langle(), (x_o), Grasp_o\rangle, \langle(x_o, x_{o'}), (x_q), Kin_o\rangle | o \in \mathcal{O}\}$$

The sampling network satisfies the graph theoretic conditions in Theorem 1. In addition, each conditional constraint manifold has full dimensionality in its input parameter space. Here Var_q, $Stable_o$, and $Grasp_o$ have no input parameters and therefore trivially have full input dimensionality. The projection $\text{proj}_{x_o, x_{o'}}(Kin_o)$ has full dimensionality under the assumption that the robot gripper's workspace has positive measure in SE(3). We consider a manifold subset of Kin_o that satisfies the submersion conditions by omitting kinematic singularities. As before, $\text{proj}_{x_q, x_{q'}}(Motion)$ has full input dimensionality.

This sampling network structure generalizes to all pick-and-place problems with goal constraints on object poses and robot configurations. Solutions are alternating sequences of *Move*, *Pick*, *MoveH*, and *Place* transitions where *Move* and *MoveH* may be repeated zero to arbitrarily many times. Each new cycle introduces a new grasp parameter, pose parameter, two configuration parameters, and two trajectory parameters. However, the only interaction with the next cycle is through the beginning and ending configurations that serve as the input parameters for the next move transition. Thus, this small set of conditional constraint manifolds is enough to define sample spaces for a large set of pick-and-place problems involving many objects.

To better visualize a pick-and-place sample space, we investigate a one-dimensional example where $\mathcal{Q}, \mathcal{S}_A$, $\mathcal{S}_B \subset \mathbb{R}$. Figure 10 fixes the plan skeleton $\vec{a} = (MoveH^A, Place^A)$ and investigates robustness properties of three problems varying the initial pose b_0 of block B. The robot starts off holding block A with grasp a_0, so in the initial state, $x_h^0 = A$ and $x_A^0 = a_0$. The robot may only grasp block A when A touches its left or right side. Therefore, the set of grasps $\mathcal{G}_A = \{-a_0, a_0\}$ is finite. In addition, the kinematic constraint on (x_o, x_o', x_q) results in a plane within $\mathbb{R}^3$.

$$Kin_o = \langle(x_o', x_o, x_q), \{q + g = p | (g, p, q) \in \mathbb{R}^3\}\rangle$$

The goal constraint $\mathcal{C}_* = \{Region_A\}$ requires that A be placed within a goal region. The plan skeleton has the following free parameters: x_q^1 is the final robot configuration, u_t^1 is the $MoveH^A$ trajectory, and x_A^2 is the final pose of block A. Once again, u_t^1 can be uniquely determined given x_q^1. The choices of these parameters must satisfy the following constraints:

$$\{Motion(q_0, u_t^1, x_q^1), CFreeH(u_t^1, a_0),$$
$$CFreeH(u_t^1, a_0, b_0), Grasp(a_0),$$
$$Stable(x_A^2), Kin(a_0, x_A^2, x_q^1), Region(x_A^2)\}$$

Varying b_0 only affects the $CFreeH(u_t^1, a_0, b_0)$ constraint. The sample space S is the one-dimensional manifold given by $Kin(a_0, x_A^2, x_q^1)$.

Each plot visualizes values of (x_A^2, x_q^1) that satisfy the Kin_A (blue line), $Region_A$ (red rectangle), and $CFreeH_{A,B}$ (green rectangle) constraints. For this example, we use x_q^1 as a surrogate for u_t^1 with respect to $CFreeH(u_t^1, a_0, b_0)$. The yellow region indicates values of (x_A^2, x_q^1) that satisfy the all the constraints and therefore result in a plan. Problem 1 is unsatisfiable because the constraints have no intersection. Problem 2 has only a single plan and therefore has zero measure with respect to S. Owing to this, Problem 2 is satisfiable but not robustly satisfiable. Problem 3 has a one-dimensional interval of plans on S, so it is robustly satisfiable. However, this set of solutions has zero measure with respect to $\mathbb{R}^2$. Without the identification of the sample space S, this problem would not be deemed robustly satisfiable.

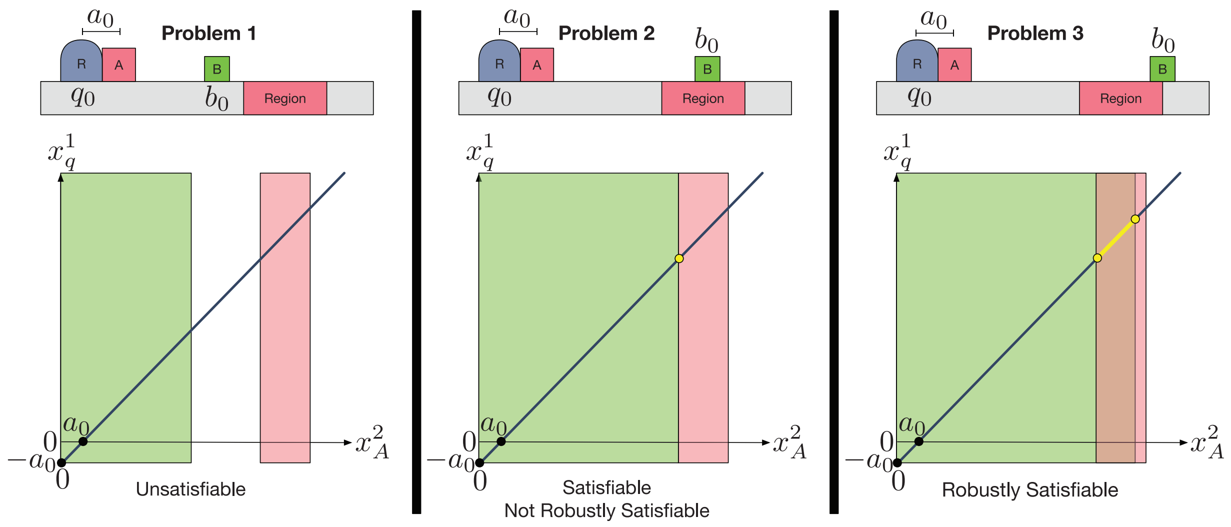


Fig. 10. From left to right: Unsatisfiable, satisfiable (but not robustly satisfiable), and robustly satisfiable pick-and-place problem for plan skeleton $\vec{a} = (MoveH^A, Place^A)$. The top row displays each one-dimensional problem varying b_0. The bottom row visualizes the plan parameter-space for free parameters x_A^2, x_q^1. Yellow regions of plan parameter space correspond to pairs of x_A^2, x_q^1 satisfying the plan skeleton.

6. Conditional samplers

Now that we have identified sample spaces that arise from dimensionality-reducing constraints, we can design samplers to draw values from these spaces. Traditional samplers either deterministically or non-deterministically draw a sequence of values $s = (v_p^1, v_p^2, \ldots)$ from the domain $\mathcal{Z}_p$ of a single parameter p. To solve problems involving dimensionality-reducing constraints using sampling, we must extend this paradigm in two ways. First, we must intentionally design samplers that draw values of several variables involved in one or more dimensionality-reducing constraints. Second, we need to construct samplers conditioned on particular values of other variables to sample values at the intersection of several dimensionality-reducing constraints. Thus, our conditional treatment of samplers will closely mirror the treatment of constraints.

Definition 19. *A conditional sampler $\psi = \langle I, O, \mathcal{C}, f \rangle$ is given by a function $f(\bar{v}_I) = (\bar{v}_O^1, \bar{v}_O^2, \ldots)$ from input values $\bar{v}_I$ for parameter indices I to a sequence of output values $\bar{v}_O$ for parameter indices O. The graph of f satisfies a set of constraints $\mathcal{C}$ on $I \cup O$.*

We will call any ψ with no inputs $I = ()$ an *unconditional sampler*. The graph implicitly contains output variable domain constraints $Var_p = \langle (z_p), \mathcal{Z}_p \rangle$ for $p \in O$. The function f may produce sequences that are enumerably infinite or finite. Let $\text{NEXT}(f(\bar{v}_I)) = \bar{v}_O$ produce the next output values in the sequence. The function f may be implemented to non-deterministically produce a sequence using random sampling. It is helpful algorithmically to reason with conditional samplers for particular input values.

Definition 20. A conditional sampler *instance* $s = \psi(\bar{v}_I)$ is a conditional sampler paired with input values $\bar{v}_I$.

We typically design conditional samplers to draw values from the conditional constraint manifolds present within a domain. For example, consider the conditional sampler ψ_{IK}^o for the kinematic constraint Kin_o.

$$\psi_{IK}^o = \langle (x_o, x_o'), (x_q), \{Kin_o\}, \text{INVERSE-KIN} \rangle$$

Here ψ_{IK}^o has input parameters $I = (x_o, x_o')$ and output parameters $O = (x_q)$. Its function $f = \text{INVERSE} - \text{KIN}$ performs inverse kinematics, producing configurations q that have end-effector transform pg^{-1} for world pose p and grasp pose g. For a seven-degree-of-freedom manipulator in SE(3), this would sample from a one-dimensional manifold.

Like conditional constraints, conditional samplers can be composed in a *sampler sequence* $\vec{\psi} = (\psi_1, \ldots, \psi_k)$ to produce a vector of values for several parameters jointly. A well-formed sampler sequence for a set of parameters Θ satisfies $\Theta = \bigcup_{j=1}^n O_j$ as well the conditions from Theorem 1. Each parameter must be an output of exactly one conditional sampler and later conditional samplers must only depend on earlier samplers. The set of values generated by the sampler sequence $\vec{\psi}$ is given by

$$F(\vec{\psi}) = \{\vec{v} | \vec{v}_{O_1} \in f_1(), \vec{v}_{O_2} \in f_2(\vec{v}_{I_2}), \ldots, \vec{v}_{O_k} \in f_k(\vec{v}_{I_k})\}$$

We are interested in identifying combinations of conditional samplers that will provably produce a solution for robustly feasible problems. Similar to $\mathcal{M}_{\mathcal{C}}$, for a set of constraints $\mathcal{C}$ and set of conditional samplers Ψ, let $\Psi_{\mathcal{C}}$ be the set of conditional samplers appropriate for each constraint.

Definition 21. A set of conditional samplers Ψ is *sufficient* for a robustly satisfiable plan skeleton $\vec{a}$ with respect to conditional constraint manifolds $\mathcal{M}$ if there exists a sampler sequence $\vec{\psi} \subseteq \Psi_{\mathcal{C}_{\vec{a}}}$ such that $F(\vec{\psi}) \cap \mathcal{C}_{\vec{a}} \neq \emptyset$ with probability one.

Definition 22. A set of conditional samplers Ψ is *sufficient* for a domain $\langle \mathcal{D}, \mathcal{M} \rangle$ if for all robustly feasible $\mathcal{P} \in \mathcal{D}$, there exists a robustly satisfiable plan skeleton $\vec{a}$ for which Ψ is sufficient.

In practice, the set of conditional samplers $\Psi(\mathcal{S})$ is often a function of the common transition system $\mathcal{S}$ for a domain $\langle \mathcal{D}, \mathcal{M} \rangle$, generating a different set of conditional samplers Ψ per domain. Alternatively, the transition system $\mathcal{S}$ can be thought of as a meta-parameter for each $\psi \in \Psi$. For example, a robot configuration conditional sampler can be automatically constructed from the number of joints and joint limits within a transition system $\mathcal{S}$. Similarly, a stable object placement conditional sampler can be parameterized by the stable surfaces defining the Stable constraint for a particular $\mathcal{S}$.

A conditional sampler must generally produce values covering its constraint manifold to guarantee completeness across a domain. This ensures that the sampler can produce values within every neighborhood on the constraint manifold. This property is advantageous because it is robust to other adversarial, worst-case constraints that only admit solutions for a neighborhood of values on the constraint manifold that the sampler is unable to reach. In motion planning, a traditional sampler $s = (v^i)_{i=1}^{\infty}, \ldots$ is *dense* with respect a topological space Z if the topological closure of its output sequence is Z (LaValle, 2006). The *topological closure* of s is the union of s and its *limit points*, points $z \in Z$ for which every neighborhood of z contains a point in s. We extend this idea to conditional samplers for conditional constraint manifolds.

Definition 23. A conditional sampler $\psi = \langle I, O, \mathcal{C}, f \rangle$ is *dense* with respect to a conditional constraint manifold $\langle I, O, M \rangle$ if $\forall \bar{v}_I \in \text{proj}_I(M)$, $f(\bar{v}_I)$ is dense in $\text{proj}_O(\text{proj}_I^{-1}(\bar{v}_I))$ with probability one.

This definition encompasses a large family of deterministic and non-deterministic sampling strategies including sampling $\text{proj}_O(\text{proj}_I^{-1}(\bar{v}_I))$ uniformly at random and independently. The following theorem indicates that dense conditional samplers for the appropriate conditional constraints will result in a sufficient collection of samplers. Thus, a set of dense conditional samplers for individual conditional constraint manifolds can be leveraged to be sufficient for any robustly satisfiable plan skeleton.

Theorem 2. *A set of conditional samplers Ψ is sufficient for a domain $\langle \mathcal{D}, \mathcal{M} \rangle$ if for each conditional constraint manifold $\langle I, O, M \rangle \in \mathcal{M}$, there exists a conditional sampler $\psi \in \Psi$ that is dense for $\langle I, O, M \rangle$.*

Proof. Consider any robustly feasible $\mathcal{P} \in \mathcal{D}$ and by Definition 18 any robustly satisfiable $\vec{a}$ plan skeleton for $\mathcal{P}$. By Definition 16, there exists a sample space $S = \bigcap_{i=1}^{n} \widehat{M}_i$ formed from conditional constraint manifolds $\{\langle I_1, O_1, M_1 \rangle, \ldots, \langle I_n, O_n, M_n \rangle\} \subseteq \mathcal{C}_{\mathcal{M}}$ and a neighborhood of parameter values $\mathcal{N}(\vec{z}) \subseteq S$ satisfying $\mathcal{C}_{\vec{a}}$. Consider each i iteration in Theorem 1. Let $\bar{y}_i \in \mathbb{R}^{d_i}$ be the coordinates introduced on the ith projection preimage where $d_i = \dim M_i - \dim \bar{\mathcal{Z}}_{I_i}$. Consider the atlas for S constructed by concatenating combinations of the charts for each projection preimage in the sequence. There exists an open set in the coordinate space of S centered around $\vec{z}$ that satisfies $\mathcal{C}_{\vec{a}}$. Consider a subset of the coordinate space of $\mathcal{N}(\vec{z})$ given as $\bar{Y}_1 \times \cdots \times \bar{Y}_n \subset \mathbb{R}^{\dim S}$ where each $\bar{Y}_i \subset \mathbb{R}^{d_i}$ is an open set. Any combination of values $(\bar{y}_1, \ldots, \bar{y}_n)$ where $\bar{y}_i \in \bar{Y}_i$ is contained within this set.

Consider a procedure for sampling $(\bar{y}_1, \ldots, \bar{y}_n)$ that chooses values for $\bar{y}_i$ in a progression of i iterations. On iteration i, the value of $\bar{z}_{\Theta_{i-1}}$ is fixed from the choices of $\bar{y}_1, \ldots, \bar{y}_{i-1}$ on previous iterations. Consider the submanifold defined by the projection preimage $\text{proj}_{I_i}^{-1}(\bar{z}_{I_i})$ for the ith conditional constraint manifold $\langle I_i, O_i, M_i \rangle$. Recall that $I_i \subseteq \Theta_{i-1}$. By assumption, there exists a conditional sampler $\psi \in \Psi$ that is dense for $\langle I, O, M \rangle$. Thus, $\psi(\bar{z}_{I_i})$ densely samples $\text{proj}_{I_i}^{-1}(\bar{z}_{I_i})$ producing output values $\bar{z}_{O_i}$. Correspondingly, $\psi(\bar{z}_{I_i})$ densely samples the coordinate space of $\bar{y}_i$. Upon producing $\bar{y}_i \in \bar{Y}_i$, the procedure moves to the next iteration. Because $\bar{Y}_i$ is open within $\mathbb{R}^{d_i}$ and the sampling is dense, the sampler will produce a satisfying coordinate values $\bar{y}_i$ within a finite number of steps with probability one. After the nth iteration, $\bar{z}$ given by coordinates $(\bar{y}_1, \ldots, \bar{y}_n) \in \bar{Y}_1 \times \cdots \times \bar{Y}_n$ will satisfy constraints $\mathcal{C}_{\vec{a}}$. □

6.1. Motion planning samplers

To apply Theorem 2 to the sampling network (Figure 7), we require conditional sampler for the Var_q and *Motion* conditional constraint manifolds. For Var_q, we provide an unconditional sampler ψ_Q that is equivalent to a traditional configuration sampler in motion planning.

$$\psi_Q = \langle (), (x_q), \{Var_q\}, \text{SAMPLE-CONF} \rangle$$

Its function SAMPLE-CONF densely samples $\mathcal{Q}$ using traditional methods (LaValle, 2006). For example, one implementation of SAMPLE-CONF non-deterministically samples $\mathcal{Q}$ uniformly at random and independently. For *Motion*, we provide a conditional sampler ψ_T that simply computes the straight-line trajectory between two configurations:

$$\psi_T = \langle (x_q, x_{q'}), (u_t), \{Motion\}, \text{STRAIGHT-LINE} \rangle$$

Here STRAIGHT-LINE generates a sequence with just a single value corresponding to the trajectory t between q and q'.

Because our motion planning transition system exhibits little factoring, ψ_Q samples entire states $(x_q) = \bar{x}$. In addition, each trajectory $(u_t) = \bar{u}$ computed by ψ_T corresponds to either a single transition when $u_t \in CFree$ or otherwise no transitions. Thus, ψ_Q and ψ_T can be seen as directly sampling the entire state and control spaces.

6.2. Pick-and-place samplers

In addition to $\psi^o_{IK}, \psi_Q, \psi_T$, the pick-and-place sampling network (Figure 9) requires the following unconditional samplers ψ^o_G and ψ^o_P for the Grasp and Stable constraints:

$$\psi^o_G = \langle \emptyset, (o), \{Grasp_o\}, \textsc{sample-grasp} \rangle$$
$$\psi^o_P = \langle \emptyset, (o), \{Stable_o\}, \textsc{sample-pose} \rangle$$

The implementation of SAMPLE-GRASP depends on the set of available grasps $\mathcal{G}_o$. In our experiments, $\mathcal{G}_o$ is a finite set. SAMPLE-POSE will typically sample (x, y, θ) two-dimensional pose values in the coordinate frame of each surface and returns the poses in the world frame.

In contrast to the motion planning transition system, the pick-and-place transition system exhibits substantial factoring. Factoring provides several benefits with respect to sampling. First, by exposing subsets of the states and controls, factoring enables the design of samplers that affect only several variables and constraints. Otherwise, one would require specifying samplers that produce full states $\bar{x}$ as is typically done in multi-modal motion planning (Hauser and Latombe, 2010; Hauser and Ng-Thow-Hing, 2011; Vega-Brown and Roy, 2016).

Second, factoring can improve sample efficiency. A small set of samples for each state variable X_i can lead to a large set of possible states $\bar{X} = X_1 \times \cdots \times X_n$ because of the decomposition of the state space into domains of individual variables. Consider Figure 11, which displays 10 samples for X_q, 4 samples for X_A, and 4 samples for X_B. From only 16 total samples, we arrive at 160 possible arrangements of the robot and objects. Although not all of these states are reachable, this large number of states presents more opportunities for solutions using these samples. Factoring can improve sample efficiency with respect to control samples as well. Consider the roadmap of control trajectories in the right frame of Figure 11. The 26 trajectories (13 edges) result in possibly 208 *Move* transitions owing to 16 possible arrangements of A and B.

7. Algorithms

We have shown that, given a plan skeleton and sampling network, we can construct samplers that produce satisfying values. However, the input for a factored planning problem is just a transition system, initial set of states, and goal set of states. Thus, algorithms must search over plan skeletons and sampler sequences to produce solutions. The concept of probabilistic completeness and the identification of probabilistically complete algorithms has been important for sampling-based motion planning. We extend those ideas to sampling-based planning for factored transition systems.

Definition 24. *An algorithm is probabilistically complete with respect to a domain $\langle \mathcal{D}, \mathcal{M} \rangle$ if for all robustly feasible problems $\mathcal{P} \in \mathcal{D}$, it will return a plan in finite time with probability one.*

We present algorithms that take as an input a set of conditional samplers Ψ for the domain. The algorithms are therefore *domain-independent* because the problem-specific knowledge is restricted to the constraints and samplers. We will show that these algorithms are probabilistically complete, given a set of sufficient conditional samplers Ψ for conditional constraint manifolds $\mathcal{M}$. Thus, the completeness of the resulting algorithm is entirely dependent on the conditionals samplers. We give two algorithms, INCREMENTAL and FOCUSED.

7.1. Incremental algorithm

The incremental algorithm alternates between generating samples and checking whether the current set of samples admits a solution. It can be seen as a generalization of the PRM for motion planning and the iterative FFRob algorithm for task and motion planning (Garrett et al., 2017a), which both alternate between exhaustive sampling and search phases for their respective problem classes.

The pseudocode for the incremental algorithm is displayed in Figure 12. INCREMENTAL updates a set *elements* containing certified constraint elements $C(\bar{v}_P)$ where $C = \langle P, R \rangle$ and $\bar{v}_P \in R$. Intuitively, each constraint element $C(\bar{v}_P) \in$ *elements* is a tuple of samples $\bar{v}_P$ that have been identified by INCREMENTAL to satisfy constraint C. As a result, *elements* encodes the current discretization of the transition relation T corresponding to problem $\mathcal{P}$. Namely, for a state and control triple $(\bar{x}, \bar{u}, \bar{x}') = \bar{v}$ and a clause $\mathcal{C}_a$,

$$\{C(\bar{v}_P) | C = \langle P, R \rangle \in \mathcal{C}_a\} \subseteq elements \Rightarrow \bar{v} = (\bar{x}, \bar{u}, \bar{x}') \in T$$

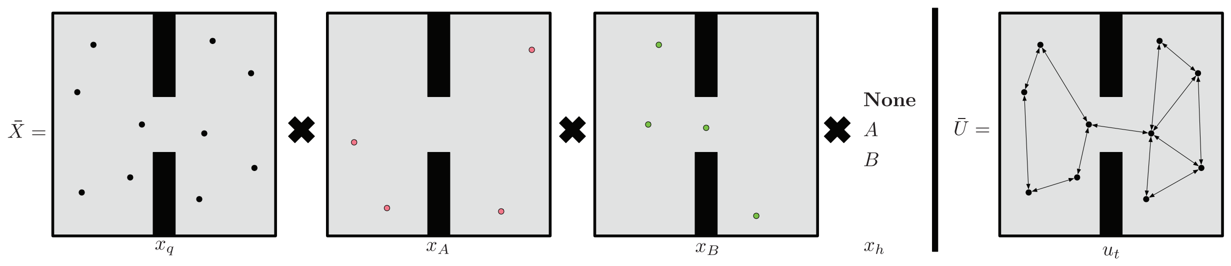


Fig. 11. A discretized state space $\bar{X}$ and control space $\bar{U}$ for a pick-and-place problem.

```
INCREMENTAL($\mathcal{P}$; $\Psi$, DISCRETE-SEARCH):
1  $elements$ = INITIAL-ELEMENTS($\mathcal{P}$)
2  $queue$ = INSTANTIATE-SAMPLERS($elements$; $\Psi$)
3  while True:
4      $\langle \vec{a}, \vec{x}, \vec{u} \rangle$ = DISCRETE-SEARCH($\mathcal{P}$, $elements$)
5      if $\vec{a} \neq$ None:
6          return $\vec{u}$
7      $processed = \emptyset$
8      PROCESS-SAMPLERS($queue$, $processed$, $elements$;
                                   SAMPLE, len($queue$))
9      PUSH($queue$, $processed$)
```

Fig. 12. The pseudocode for the incremental algorithm.

```
SAMPLE($s$):
1  $\psi(\bar{v}_I) = s$; $\langle I, O, \mathcal{C}, f \rangle = \psi$
2  $\bar{v}_O$ = next($f(\bar{v}_I)$)
3  if $\bar{v}_O$ = None:
4      return {}
5  return $\{C(\bar{v}_I + \bar{v}_O) \mid C \in \mathcal{C}\}$

INSTANTIATE-SAMPLERS($elements$; $\Psi$):
1  $samples = \{v \mid C(\bar{v}) \in elements, v \in \bar{v}\}$
2  $instances = \emptyset$
3  for $\psi = \langle I, O, \mathcal{C}, f \rangle$ in $\Psi$:
4      for $\bar{v}_I$ in product($samples$, $|I|$):
5          $instances$ += $\{\psi(\bar{v}_I)\}$
6  return $instances$

PROCESS-SAMPLERS($queue$, $processed$, $elements$; PROCESS, $k$):
1  while (len($queue$) $\neq 0$) and (len($processed$) $< k$) :
2      $s$ = POP($queue$)
3      $elements$ += PROCESS($s$)
4      for $s'$ in INSTANTIATE-SAMPLERS($elements$; $\Psi$):
5          if $s'$ not in ($queue$ + $processed$):
6              PUSH($queue$, $s'$)
7      $processed$ += $\{s\}$
```

Figure 13. The pseudocode for the procedures shared by both the incremental and focused algorithms.

As INCREMENTAL identifies new constraint *elements* and adds them to elements, it in turn identifies additional possible transitions.

In addition, INCREMENTAL maintains *queue*, a first-in–first-out queue of sampler instances. The procedure INITIAL-ELEMENTS gives the set of elements resulting from initial samples present in problem $\mathcal{P}$. Define an *iteration* of INCREMENTAL to be the set of commands in body of the **while** loop. On each iteration, INCREMENTAL first calls DISCRETE-SEARCH to attempt to find a plan $\langle \vec{a}, \vec{x}, \vec{u} \rangle$ using *elements*. The procedure DISCRETE-SEARCH searches the current discretization of problem $\mathcal{P}$ given by *elements*. We assume DISCRETE-SEARCH is any sound and complete discrete search algorithm such as a breadth-first search (BFS). We outline several possible implementations of DISCRETE-SEARCH in Section 8. If DISCRETE-SEARCH is successful, the sequence of control inputs $\vec{u}$ is returned. Otherwise, DISCRETE-SEARCH produces $\vec{a} =$ **None**. In this case, INCREMENTAL calls the PROCESS-SAMPLERS subroutine to sample values from at most **len**(*queue*) sampler instances using the function SAMPLE. The procedure SAMPLE in Figure 13 has a single sampler-instance argument s and queries the **next** set of output values $\bar{v}_O$ in the sequence $f(\bar{v}_I)$. If the sequence has not been enumerated, i.e. NEXT$(f(\bar{v}_I)) \neq$ **None**, it returns a set of constraint elements $C(\bar{v}_I + \bar{v}_O)$ that values $\bar{v}_I + \bar{v}_O$ satisfy together.

The procedure PROCESS-SAMPLERS iteratively instantiates and processes sampler instances s. Its inputs are a queue of sampler instances, a set of already *processed* sampler instances, the set of constraint elements, and two additional parameters that are used differently by INCREMENTAL and FOCUSED: the procedure PROCESS $\in$ {SAMPLE, SAMPLE-LAZY} takes as input a sampler instance and returns a set of elements, and k is the maximum number of sampler instances to process. On each iteration, PROCESS-SAMPLERS pops a sampler instance s off of *queue*, adds the result of PROCESS to *elements*, and adds s to *processed*. Constraints specified using tests, such as collision constraints, are immediately evaluated upon receiving new values. The procedure INSTANTIATE-SAMPLERS produces the set of sampler instances of samplers Ψ formed from constraint elements *elements*. For each sampler ψ, INSTANTIATE-SAMPLERS creates a sampler instance $s = \psi(\bar{v}_I)$ for every combination of values $\bar{v}_I$ for the input parameters of ψ that satisfy the input variable domain constraints. New, unprocessed sampler instances s' resulting from the produced *elements* are added to *queue*. PROCESS-SAMPLERS terminates after k iterations or when *queue* is empty. Afterwards, INCREMENTAL adds the *processed* sampler instances back to *queue* to be used again on later iterations. This ensures each sampler instance is revisited arbitrarily many times.

Theorem 3. INCREMENTAL *is probabilistically complete for a domain* $\langle \mathcal{D}, \mathcal{M} \rangle$ *given a sufficient set of conditional samplers for* $\langle \mathcal{D}, \mathcal{M} \rangle$.

Proof. Consider any robustly feasible problem $\mathcal{P} \in \mathcal{D}$. By Definitions 6 and 7, there exists a sampler sequence $\vec{\psi} = (\psi_1, \ldots, \psi_k)$ that with probability one, a finite number of calls to SAMPLE produces values that are parameters in $\mathcal{C}_{\vec{a}}$ for some robustly satisfiable plan skeleton $\vec{a}$. In its initialization, INCREMENTAL adds all sampler instances s available from the $\mathcal{P}$'s constants. On each iteration, INCREMENTAL performs SAMPLE(s) for each sampler instance s in *queue* at the start of the iteration. There are a finite number of calls to SAMPLE each iteration. The resulting constraints elements SAMPLE(s) are added to elements and all new sampler instances s' are added to *queue* to be sampled later. The output values from each sampler instance will later become input values for all other appropriate conditional samplers. This process will indirectly sample all appropriate sampler sequences including $\vec{\psi}$. Moreover, because s is re-added to *queue*, it will be revisited on each iteration. Thus, SAMPLE(s) will be computed until a solution is found. Therefore, each sampler sequence will also be sampled not only once but arbitrarily many times. INCREMENTAL will produce satisfying values from $\vec{\psi}$ within a finite number of iterations.

Because DISCRETE-SEARCH is assumed to be sound and complete, DISCRETE-SEARCH will run in finite time and return a correct plan if one exists. On the first iteration in which a solution exists within samples, DISCRETE-SEARCH will produce a plan $\vec{a} \neq$ **None**. In addition, INCREMENTAL will itself return the corresponding sequence of control inputs $\vec{u}$ as a solution. □

Because INCREMENTAL creates sampler instances exhaustively, it will produce many unnecessary samples. This results in combinatorial growth in the number of queued sampler instances as well as the size of the discretized state space. This motivates our second algorithm, which is able to guide the selection of samplers by integrating the search over structure and search over samples.

7.2. Focused algorithm

The FOCUSED algorithm uses *lazy samples* as placeholders for actual concrete sample values. Lazy samples are similar in spirit to symbolic references (Srivastava et al., 2014). The lazy samples are optimistically assumed to satisfy constraints with concrete samples and other lazy samples via *lazy constraint elements*. Lazy constraint elements produce an optimistic discretization of $\mathcal{T}$, characterizing transitions that may exist for some concrete binding of the lazy samples involved. This allows DISCRETE-SEARCH to reason about plan skeletons with some free parameters. After finding a plan, FOCUSED calls samplers that can produce values for the lazy samples used. As a result, FOCUSED is particularly efficient on easy problems in which a large set of samples satisfy each constraint. This algorithm is related to a lazy PRM (Bohlin and Kavraki, 2000; Dellin and Srinivasa, 2016), which defers collision checks until a path is found. However instead of just deferring collision checks, FOCUSED defers generation of sample values until an optimistic plan is found. In a pick-and-place application, this means lazily sampling poses, inverse kinematic solutions, and trajectories. Because FOCUSED plans using both lazy samples and concrete samples, it is able to construct plans that respect constraints on the actual sample values. In addition, by using lazy samples, it can indicate the need to produce new samples when the existing samples are insufficient.

The pseudocode for the focused algorithm is shown in Figure 14. The focused algorithm uses the same subroutines in Figure 13 as the incremental algorithm. Once again, let an *iteration* of FOCUSED be the set of commands in body of the **while** loop. Define an *episode* of FOCUSED to be the set of iterations between the last *sampled* reset and the next *sampled* reset. Let the initialization of *sampled* in line 1 also be a reset. On each iteration, the FOCUSED algorithm creates a new *queue* and calls PROCESS-SAMPLERS to produce *mixed_elements*. It passes the procedure SAMPLE-LAZY rather than SAMPLE in to PROCESS-SAMPLERS. For each output o of ψ, SAMPLE-LAZY creates a unique lazy sample l_o^ψ for the combination of ψ and o. Then, for each lazy constraint element e formed using l_o^ψ, s is recorded as the sampler instance that produces values satisfying the element using *e.instance*. For a pose sampler instance $\psi_P^o()$, SAMPLE-LAZY creates a single lazy sample l_1^P and returns a single lazy constraint element $Stable(l_1^P)$:

$$\text{LAZY-SAMPLE}(\psi_P()) = \{Stable(l_1^P)\}$$

DISCRETE-SEARCH performs its search using *mixed_elements*, a mixed set of *elements* and *lazy_elements*. If DISCRETE-SEARCH returns a plan, FOCUSED first checks whether it does not require any *lazy_elements*, in which case it returns the sequence of control inputs $\bar{u}$. Otherwise, it calls RETRACE-INSTANCES to recursively extract the set of sampler instances used to produce the *lazy_elements*. RETRACE-INSTANCES returns just the set of ancestor sampler instances that do not contain lazy samples in their inputs. For each ancestor sampler instance s, FOCUSED samples new output values and adds any new constraint elements to *new_elements*. To ensure all relevant sampler instances are fairly sampled, each s is then added to *sampled*. This prevents these sampler instances from constructing lazy samples within PROCESS-SAMPLERS on subsequent iterations. In addition, elements are added to *new_elements* before they are moved to elements to limit the growth in sampler instances. When DISCRETE-SEARCH fails to find a plan, *new_samples* are added to *elements, sampled* is reset, and this process repeats on the next episode.

While not displayed in the pseudocode, the focused algorithm has the capacity to identify infeasibility for some problems. When DISCRETE-SEARCH fails to find a plan and *sampled* is empty, the problem is infeasible because the discretized problem with optimistic lazy samples is infeasible. If no graph on conditional samplers Ψ contains cycles, then a lazy sample can be created for each sampler instance rather than each sampler. Then, RETRACE-INSTANCES can sample values for lazy elements that depend on the values of other lazy elements. Finally, *new_elements* can be safely added directly to *elements*. These modifications can speed up planning time by requiring fewer calls to SOLVE-DISCRETE. For satisficing planning, to bias discrete-search to use few lazy samples, we add a non-negative cost to each transition instance corresponding to the number of lazy samples used and use a cost-sensitive version of SOLVE-DISCRETE. In this context SOLVE-DISCRETE can be thought of optimizing for a plan that requires the least amount of additional sampler effort. Thus, plans without lazy samples have low cost while plans with many lazy samples have high cost.

Theorem 4. FOCUSED *is probabilistically complete for a domain $\langle \mathcal{D}, \mathcal{M} \rangle$ given a sufficient set of conditional samplers for $\langle \mathcal{D}, \mathcal{M} \rangle$.*

Proof. As in Theorem 3, consider any robustly feasible problem $\mathcal{P} \in \mathcal{D}$. By Definitions 21 and 22, there exists a sampler sequence $\vec{\psi} = (\psi_1, \ldots, \psi_k)$ that with probability one, in a finite number of calls to SAMPLE produces values

```
SAMPLE-LAZY($s$):
    $\psi(\bar{v}_I) = s; \langle I, O, \mathcal{C}, f \rangle = \psi$
    $\bar{l}^{\psi}_O = (l^{\psi}_o \mid o \in O)$
    $lazy\_elements = \{C(\bar{v}_I + \bar{l}^{\psi}_O) \mid C \in \mathcal{C}\}$
    for $e \in lazy\_elements$:
        $e.instance = s$
    return $lazy\_elements$
```

```
RETRACE-INSTANCES($target\_elements$, $elements$):
    $instances = \emptyset$
    for $e$ in $(target\_elements \setminus elements)$:
        $\psi(\bar{v}_I) = e.instance$
        $ancestors$ = RETRACE-INSTANCES($\{Var(\bar{v}) \mid \bar{v} \in \bar{v}_I\}$)
        if $ancestors = \emptyset$:
            $instances$ += $\{\psi(\bar{v}_I)\}$
        $instances$ += $ancestors$
    return $instances$
```

```
FOCUSED($\mathcal{P}; \Psi$, DISCRETE-SEARCH):
    $elements$ = INITIAL-ELEMENTS($\mathcal{P}$)
    $new\_elements = \emptyset$; $sampled = \emptyset$
    while True:
        $queue$ = INSTANTIATE-SAMPLERS($elements$; $\Psi$)
        $mixed\_elements$ = copy($elements$)
        PROCESS-SAMPLERS($queue$, copy($sampled$), $mixed\_elements$;
                                    SAMPLE-LAZY, $\infty$)
        $\langle \vec{a}, \vec{x}, \vec{u} \rangle$ = DISCRETE-SEARCH($\mathcal{P}$, $mixed\_elements$)
        if $\vec{a}$ = None:
            $elements$ += $new\_elements$
            $new\_elements = \emptyset$; $sampled = \emptyset$
            continue
        $plan\_elements = \{C(\vec{x} + \vec{u}) \mid C \in \mathcal{C}_{\vec{a}}\}$
        if $plan\_elements \subseteq elements$:
            return $\vec{u}$
        for $s$ in RETRACE-INSTANCES($plan\_elements$, $elements$):
            $new\_elements$ += SAMPLE($s$)
            $sampled$ += $\{s\}$
```

Fig. 14. The pseudocode for the focused algorithm.

that are parameters in $\mathcal{C}_{\vec{a}}$ for some robustly satisfiable plan skeleton $\vec{a}$. At the start of an episode, *elements* implicitly represents a set of partially computed sampler sequences. We show that between each episode, for each partially computed sampler sequence that corresponds to some plan skeleton, a next sampler in the sampler sequence will be called. In addition, both the new partial sampler sequence as well as the old one will be present within *elements* during the next episode.

On each iteration, FOCUSED calls DISCRETE-SEARCH to find a plan that uses both real samples and lazy samples. FOCUSED calls SAMPLE for each sampler instance s corresponding to a lazy sample along the plan. In addition, by adding s to *sampled*, FOCUSED prevents the lazy samples resulting from s from being used for any future iteration within this episode. This also prevents DISCRETE-SEARCH from returning the same plan for any future iteration in this episode. The set *elements* is fixed for each episode because new samples are added to *new_elements* rather than *elements*. Thus, there are a finite number of plans possible within each episode. In addition, the number of iterations within the episode is upper bounded by the initial number of plans. Each plan will either be returned on some iteration within the episode and then be blocked or it will be incidentally blocked when SEARCH returns another plan. Either way, SAMPLE will be called for a sampler instance s on its remaining sampler sequence. When no plans remain, SEARCH will fail to find a plan. Then, focused resets, allowing each $s \in$ *sampled* to be used again, and the next episode begins.

Because each episode calls SAMPLE for at least one ψ along each possible partial sampler sequence, $\vec{\psi}$ will be fully sampled once after at most k episodes. Moreover, each subsequent episode will sample $\vec{\psi}$ again as new partial sampler sequences are fully computed. Thus, $\vec{\psi}$ will be fully sampled arbitrarily many times. Consider the first episode in which a solution exists within *elements*. DISCRETE-SEARCH is guaranteed to return a plan only using only *elements* within this episode. This will happen, at latest, when all plans using lazy elements are blocked by *sampled*. Then, FOCUSED will itself return the corresponding sequence of control inputs as a solution. □

It is possible to merge the behaviors of the incremental and focused algorithms and toggle whether to eagerly or lazily SAMPLE per conditional sampler. This allows inexpensive conditional samplers to be immediately evaluated while deferring sampling of expensive conditional samplers. This fusion leads to a variant of the FOCUSED algorithm where sampler instances switch from being lazily evaluated to eagerly evaluated when they are added to *sampled*. In this case, *sampled* need not be reset upon DISCRETE-SEARCH failing to identify a plan.

8. Discrete search

The procedure DISCRETE-SEARCH takes as input a factored transition problem $\mathcal{P}$ and a set of constraint elements *elements*. The set of constraint elements *elements* is used to derive *transitions*, a discretization of transition relation $\mathcal{T}$ for problem $\mathcal{P}$. This is done by first extracting the discretized variable domain Z_p for each parameter index p:

$$Z_p = \{\bar{v}_p | \exists\ C(\bar{v}_P) \in elements.\ p \in P\}$$

The discretized variable domains result a discretized state space $\bar{X} = X_1 \times \cdots \times X_m$ and control space $\bar{U} = U_1 \times \cdots \times U_n$. The discretized set of transitions is then

$$transitions = \{(\bar{x}, \bar{u}, \bar{x}') \in \bar{X} \times \bar{Z} \times \bar{X} | \exists\ \mathcal{C}_a,.\ \forall\ C = \langle P, R \rangle \in \mathcal{C}_a.\ C(\bar{z}_P) \in elements\}$$

A straightforward implementation of DISCRETE-SEARCH is a BFS from $\bar{x}_0$ using *transitions* to define the set of directed edges $\{(\bar{x}, \bar{x}')|(\bar{x}, \bar{u}, \bar{x}') \in transitions\}$ defined on vertices $\bar{X}$. Note that the control samples $\bar{u}$ are used to identify transitions, but play no role in the BFS itself. As an optimization, the outgoing edges from a state $\bar{x}$ can be dynamically computed by considering each clause $\mathcal{C}_a$, substituting the current values for $\bar{x}$, and identifying all combinations of $\bar{u}$ and $\bar{x}'$ resulting in $(\bar{x}, \bar{u}, \bar{x}') \in transitions$. This can be further optimized by fixing the values of any $\bar{u}$, $\bar{x}'$ constrained by equality. Whereas a BFS avoids explicitly constructing the full discretized state-space, it will still search the entire state-space reachable from $\bar{x}_0$ in fewer transitions than the length of the shortest plan. This can be prohibitively expensive for problems with significant factoring such as pick-and-place problems where many choices of objects to manipulate result in a large branching factor.

8.1. Factored planning

The artificial intelligence community has developed many algorithms that are much more efficient than classical graph search algorithms for high-dimensional, factored problems. These algorithms exploit both the factored state representation and transitions with many equality constraints to guide search using domain-independent heuristics. Many heuristics are derived by solving an easier approximation of the original search problem. This leads to both admissible (Bonet and Geffner, 2001) and empirically effective heuristics (Helmert, 2006; Hoffmann and Nebel, 2001). These heuristics can frequently avoid exploring most of the discrete state space and even efficiently identify many infeasible problem instances.

In our experiments, we use the efficient FastDownward planning toolkit (Helmert, 2006), which contains implementations of many of these algorithms. FastDownward, as well as many other planners, operate on states described as a finite set of discrete variables. For example, the foundational STRIPS (Fikes and Nilsson, 1971) planning formalism uses binary variables in the form of logical propositions. We instead consider the Simplified Action Structures (SAS +) (Bäckström and Nebel, 1995) planning formalism, which allows variables with arbitrary finite domains. This allows a factored transition system state $\bar{x}$ to also be a legal SAS + state where each state variable x_p has a finite discretized domain X_p. Discrete transitions in SAS + are described using a precondition and effect action model.

Definition 25. An *action* $\langle pre, eff\rangle$ is given by sets of constant equality conditions *pre* on $\bar{x}$ and eff on $\bar{x}'$. State variables i omitted from *eff* are assumed to be constrained by pairwise equality constraints $x_i = x_{i'}$.

A single action will represent many different transitions if some state variables are not mentioned within *pre*. Thus, action models can be advantageous because they compactly describe many transitions using a small set of actions.

To use these algorithms, we automatically compile *transitions* into SAS + . We could instead automatically compile to Planning Domain Definition Language (PDDL) (Edelkamp, 2004; McDermott et al., 1998), a standardized artificial intelligence planning language used in competitions. However, many PDDL planners first compile problem instances into a formalism similar to SAS + (Helmert, 2006), so we directly use to this representation.

8.2. Action compilation

A factored transition system with a discretized set of constraint elements can be compiled into SAS + as follows. First, the goal constraints $\mathcal{C}_*$ are converted into a "goal transition"

$$\mathcal{C}_* \cup \{x'_{goal} = \textbf{True}\} \cup \{x_1 = x_{1'}, \ldots, x_m = x_{m'}\}$$

by adding a state variable x_{goal} that is true when the goal constraints have been satisfied. This allows the new goal $\mathcal{C}_{*'} = \{x'_{goal} = \textbf{True}\}$ to be represented with a single equality constraint. Owing to this additional state variable, all existing transitions are augmented with an equality constraint $\{x_{goal} = x'_{goal}\}$.

Each clause $\mathcal{C}_a$ is compiled into a set of actions by first identifying its set of possible parameters P_a, which is composed of each x_i, u_j present within $C \in \mathcal{C}_a$ as well as all of $\bar{x}'$. The inclusion of the entirely of $\bar{x}'$ reflects that, after applying an action, each state variable may change. Many clauses $\mathcal{C}_a$ contain constant and pairwise equality constraints that fully constrain some parameters. Thus, the subset of free parameters $F_a \subseteq P_a$ is determined by defining a graph on parameters and samples where undirected edges are pairwise equality constraints. Connected components in the graph are parameters and samples that are transitively constrained by equality. For each connected component that does not contain sample, a single parameter $f \in P_a$ is selected to represent the component. Finally, the clause is grounded by considering every binding of the free parameters F_a satisfying $\mathcal{C}_a$. This creates ground action $\langle pre, eff\rangle$ for each binding where *pre* contains an equality constraint from each $x_i \in P_a$ to the bound value of its corresponding free parameter $f \in F_a$ and *eff* contains an equality constraint from each $x_{i'}$ to the bound value of its corresponding free parameter $f \in F_a$ if $[x_i = x_{i'}] \notin \mathcal{C}_a$. Consider the following actions generated for $\mathcal{C}^o_{Pick}$:

$$\{\langle pre = \{x_o = p, x_h = \textbf{None}, x_q = q\},$$
$$eff = \{x_o = g, x_h = o\}\rangle | \exists g, p, q.\ Kin_o(g, p, q) \in elements\}$$

Here $\mathcal{C}^o_{Pick}$ and $\mathcal{C}^o_{Place}$ only have three free parameters, so $F^o_{Pick} = \{p, g, q\}$. This results in a compact descriptions of their transitions. Now consider the actions generated for $\mathcal{C}_{Move}$:

$$\{\langle pre = \{x_q = q, x_h = \textbf{None}\} \cup \{x_o = p_o | o \in \mathcal{O}\},$$
$$eff = \{x_{q'} = q'\}\rangle | \exists q, t, q', p_1, \ldots, p_{|\mathcal{O}|}.$$
$$\{Motion(q, t, q'), CFree(t)\} \cup$$
$$\{CFree_o(t, p_o) | o \in \mathcal{O}\} \subseteq elements\}$$

Here $\mathcal{C}_{Move}$ and $\mathcal{C}^o_{MoveH}$ have $3 + |\mathcal{O}|$ free parameters because $F_{Move} = \{q, t, q', p_1, \ldots, p_{|\mathcal{O}|}\}$. For non-unary discretization of each object variable, the number of transitions grows exponentially in $|\mathcal{O}|$. Despite this, each *eff* only involves one variable and there is only one control variable. The rest of the state variables are solely used to determine action feasibility. In addition, each constraint has low arity: Motion involves three parameters and each $CFree_o$ constraint only involves two parameters.

8.3. Axiom compilation

Low constraint arity allows us to further factor transitions by introducing *derived variables* (Edelkamp, 2004), variables evaluated from the core state variables $\bar{x}$ using rules known as *axioms* (Helmert, 2006). Axioms are known to be useful for compactly expressing planning problems (Thiébaux et al., 2005). Axioms have the same form $\langle pre, eff \rangle$ as actions. However, they are automatically applied upon reaching a new state in contrast to actions, which are chosen by a planner.

For each non-equality constraint $C = \langle P, R \rangle \in \mathcal{C}_a$, we compute a parameterized boolean derived variable $d_C(\bar{z}_D)$. The parameterization $D = P \backslash \bar{x}$ includes parameters for the control $\bar{u}$ and subsequent state $\bar{x}'$ but excludes the current state variables $\bar{x}$. Note that $\bar{u}$ and $\bar{x}'$ are included within D to ensure that same control and subsequent state values are considered in each derived variable precondition. An axiom $\langle pre, eff \rangle$ is computed for each constraint element $C(\bar{v}) \in elements$ where $pre = \{p : v_p | p \in (P \cap \bar{x})\}$ and $eff = \{d_C(\bar{v}_D) : \textbf{True}\}$. This allows $d_C(\bar{z}_D) = \textbf{True}$ to be substituted for C within the preconditions of any action involving C. Using this substitution, an action is performable if, for each constraint C, the values of state variables $P \cap \bar{x}$ complete a constraint element within elements. As a result, $\bar{x}$ can be removed from the possible parameters P_a of $\mathcal{C}_a$. Because each P_a now involves fewer parameters, the set of resulting actions and axioms instances is generally much smaller than before.

The axioms computed for *Motion* are

$$\begin{aligned}&\{\langle pre = \{x_q = q\},\\ &eff = \{Motion(\,\cdot\,, t, q') = \textbf{True}\}\rangle | \exists q, t, q'.\\ &Motion(q, t, q') \in elements\}\end{aligned}$$

The axioms computed for $CFree_o$ are

$$\begin{aligned}&\{pre = \langle\{x_o = p\},\\ &eff = \{CFree_o(t,\,\cdot\,) = \textbf{True}\}\rangle | \exists t, p.\\ &CFree_o(t, p) \in elements\}\end{aligned}$$

In addition, $\mathcal{C}_{Move}$ can be modified to be the following:

$$\begin{aligned}&\{\langle pre = \{Motion(\,\cdot\,, t, q') = \textbf{True}, x_h = \textbf{None}\}\\ &\cup \{CFree_o(t,\,\cdot\,) : \textbf{True} | o \in \mathcal{O}\},\\ &eff = \{x_{q'} = q\}\rangle | \exists t, q'.\\ &\{Var(t), Var(q')\} \subseteq elements\}\end{aligned}$$

The resulting *Motion* and $CFree_o$ axioms as well as $\mathcal{C}_{Move}$ actions all have three or fewer parameters. The number of actions and axioms needed to describe a pick-and-place transition is linear in $|\mathcal{O}|$ rather than exponential in $|\mathcal{O}|$.

9. Tabletop manipulation

We seek to model tabletop manipulation problems involving a manipulator attached to a movable base as a factored transition system. The previously presented pick-and-place factored transition system encompasses this application, and the specified samplers lead to probabilistically complete algorithms. However, the previous formulation leads to poor performance in practice for high-dimensional robot configuration spaces as it attempts to construct control trajectories between all pairs of robot configurations. The resulting control space is similar to a simplified probabilistic roadmap (sPRM) (Kavraki and Latombe, 1998), which is known to be inefficient for high-dimensional robot configuration spaces. Instead, we model tabletop manipulation problems as transition systems in which multi-waypoint robot trajectories u_m are control parameters. This allows us to design samplers that call efficient motion planners to produce trajectories between pairs of configurations.

Rather than specify $\mathcal{C}^o_{Pick}$ and $\mathcal{C}^o_{Place}$ transitions as instantaneous contacts with each object, we represent robot trajectories moving to, manipulating, and returning from an object as a single transition $\mathcal{C}^o_{MPick}$ or $\mathcal{C}^o_{MPlace}$:

$$\begin{aligned}&\mathcal{C}^o_{MPick} = \{Stable_o, Grasp_{o'}, Manip,\\ &x_h = \textbf{None}, x_{h'} = o,\} \cup\\ &\{x_{o'} = x'_{o'}, CFree_{o'})| o' \in \mathcal{O}, o \neq o'\}\end{aligned}$$

$$\begin{aligned}&\mathcal{C}^o_{MPlace} = \{Grasp_o, Stable'_o, Manip',\\ &x_h = o, x_{h'} = \textbf{None}\} \cup\\ &\{x_{o'} = x'_{o'}, CFree_{o'} | o' \in \mathcal{O}, o \neq o'\}\end{aligned}$$

This behavior is enforced to be a manipulation constraint *Manip* on parameters x_o, x'_o, u_m representing poses x_o, x'_o for object o and trajectory u_m. Let q_0 be the initial robot configuration and let q_{Kin} be a kinematic solution for end-effector transform $x_{o'}x_o^{-1}$ grasping object o with grasp x_o^{-1} at placement $x_{o'}$. The trajectory u_m is the concatenation of a motion plan from $q_0 \rightarrow q_{Kin}$, a grasp plan, and a motion plan $q_{Kin} \rightarrow q_0$. Both motion plans are computed to be free of collisions with fixed obstacles. In addition, one motion plan avoids collisions with o placed at $x_{o'}$, and the other avoids collisions between o held at grasp x_o and fixed obstacles. Because the robot always returns to q_0, the robot configuration need not be included as a state variable.

We structure the transition system this way based on the insight that the bulk of the robot's configuration space is unaffected by the placements of movable obstacles. The moveable obstacles mostly only prevent the safe execution of manipulator trajectories above tabletops. Rather than directly plan paths between pairs of configurations manipulating objects, we instead plan paths to a home configuration chosen arbitrarily as the initial configuration q_0. This approach guarantees the feasibility of the resulting plan while not inducing significant overhead. Shorter, direct trajectories between pairs of base configurations can later be produced when post-processing a solution. Figure 15 visualizes the set of u_m trajectories as edges in a star roadmap (Garrett et al., 2017a) with q_0 as the root.

We specify the following samplers to produce values satisfying the constraints in this transition system. The grasp sampler ψ_P^o and placement ψ_P^o sampler are the same as before. The manipulation sampler ψ_M^o is similar to ψ_{IK}^o:

$$\psi_M^o = \langle (x_o, x_o'), (u_m), \{Manip_o\}, \textsc{sample-manip}\rangle$$

The procedure SAMPLE-MANIP samples a nearby base pose via inverse reachability. From this base pose, it performs manipulator inverse kinematics to identify a grasping configuration to perform the pick or place. If SAMPLE-MANIP fails to find a kinematic solution, it samples a new base pose. Otherwise, it calls a sampling-based motion planner twice to find motion plans to this grasping configuration and back. These motion plans are computed to not be in collision with fixed obstacles or o both when it is on the table and when it is held. In addition, each call has a timeout meta parameter to ensure termination. The timeout for each sampler instance of ψ_M^o is increased after each call allowing sample-manip to have an unbounded amount of time collectively over all of its calls.

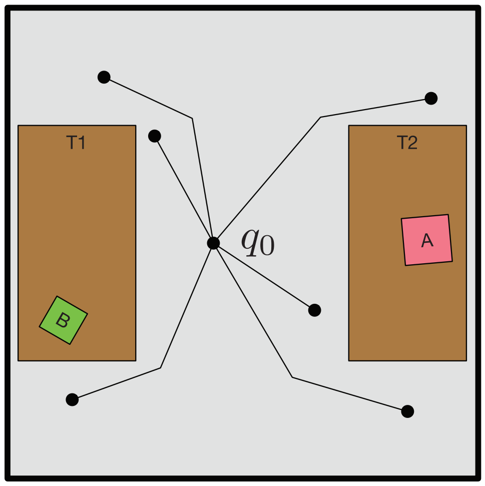


Fig. 15. Star roadmap composed of trajectories. Base configurations outside the room are identified as unreachable.

$$\vec{a}_3 = [\mathcal{C}_{MPick}^A, \mathcal{C}_{MPlace}^A]$$

$$\vec{x}_3 = [(a_0, b_0, \textbf{None}), (a_g, b_0, A), (a_1, b_0, \textbf{None})]$$

$$\vec{u}_3 = [m_1, m_3]$$

Note that the number of sampler instances sampled per iteration grows quickly. In addition, samples are generated for both objects A and B despite the task only requiring manipulating A.

10. Example mobile manipulation problem

To illustrate both the incremental and focused algorithms, we work through their steps on an example mobile manipulation problem. Consider the problem in Figure 16 with two movable objects A, B and two tables $T1, T2$. States are $\bar{x} = (x_A, x_B, x_h)$ and controls are $\bar{u} = u_m$. The initial state is $\bar{x}_0 = (a_0, b_0, \textbf{None})$ and the goal constraints are $\mathcal{C}_* = \{Region_A\}$ indicating that object A is placed on $T1$. For simplicity, we assume that each moveable object has a single grasp a_g or b_g. The values $a_0, b_0, a_g, b_g \in \text{SE}(3)$ represent continuous transformations. Similarly, manipulations m are full-body motion plans from q_0 to a grasping configuration and back. For the example, we assume that the conditional samplers never fail to produce an appropriate value.

10.1. Incremental algorithm

Table 1 traces the sampler instances S_i for which SAMPLE is called paired with the resulting element for each iteration i of the incremental algorithm. The set of *elements* available on each iteration is the union of the previously sampled elements S_j, $j < i$. The incremental algorithm fails to find a plan for two iterations and finally finds the following plan π_3 on the third iteration:

10.2. Focused example

Table 2 traces each iteration i of the incremental algorithm. We assume that *new_elements* are added directly to *elements*. As before, S_i contains the sampler instances SAMPLE paired with the resulting elements. Elements certified by *CFree* and *Region* are individually added to S_i. The set of *sampled* sampler instances are contained in S_j, $i < j$. Let $\mathcal{E}_i$ be the set of elements using lazy samples generated by PROCESS-SAMPLERS on each iteration. The union of $\mathcal{E}_i$ and elements is *mixed_elements*. The plan returned on each iteration is denoted by $\pi_i = \langle \vec{a}_i, \vec{x}_i, \vec{u}_i\rangle$. We denote the lazy sampler for each sampler as follows:

$$\textsc{lazy-sample}(\psi_G^o()) = \{Grasp(l_o^G)\}$$

$$\textsc{lazy-sample}(\psi_P^o()) = \{Stable(l_o^P)\}$$

$$\textsc{lazy-sample}(\psi_M^o(x_o, x_{o'})) = \{Manip(x_o, x_{o'}, l_m^o)\}$$

On the first iteration, the sampler instances $\psi_G^a(), \psi_P^a()$ are sampled to produce values for l_A^G, l_A^P, respectively. On the second iteration, $\psi_M^a(a_0, a_g)$ and $\psi_M^a(a_1, a_g)$ generate the manipulations required for the $\mathcal{C}_{MPick}^A$ and $\mathcal{C}_{MPlace}^A$ transitions given the new grasp a_g and placement a_1. On the final

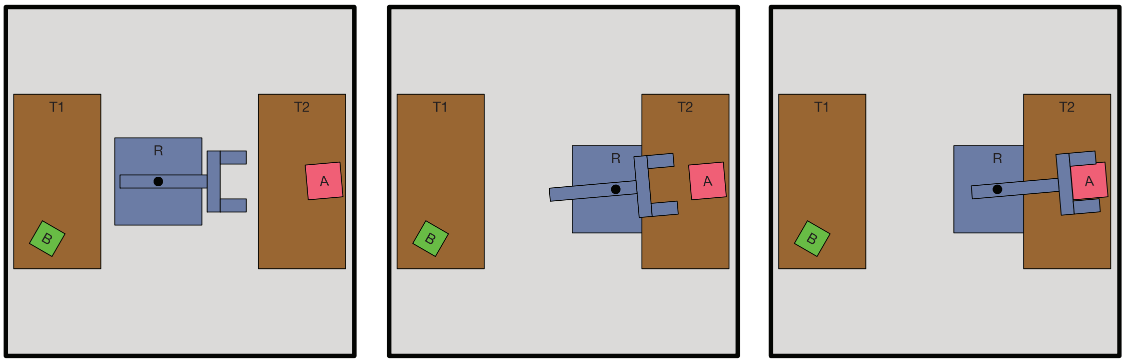


Fig. 16. Mobile manipulation example application.

Table 1. Example walkthrough of the incremental algorithm. Each S_i displays the set of sampler instances for which SAMPLE is called along with the new elements produced.

S_0	INITIAL-ELEMENTS$(\mathcal{P})$: $\{Stable(a_0), Stable(b_0))\}$
$\vec{a}_1$	**None**
S_1	$\psi^a_G() : Grasp(a_g), \psi^b_G() : Grasp(b_g), \psi^A_P() : Stable(a_1), \psi^B_P() : Stable(b_1), Region(a_1)$
$\vec{a}_2$	**None**
S_2	$\psi^A_M(a_0, a_g) : Manip(a_0, a_g, m_1), \psi^B_M(b_0, b_g) : Manip(b_0, b_g, m_2), \psi^A_M(a_1, a_g) : Manip(a_1, a_g, m_3),$ $\psi^B_M(b_1, b_g) : Manip(b_1, b_g, m_4), \psi^A_P() : Stable(a_2), \psi^B_P() : Stable(b_2), Region(a_2)$ $CFree(m_1, b_0), CFree(m_1, b_1), CFree(m_1, b_2), CFree(m_2, a_0), CFree(m_2, a_1), CFree(m_2, a_2)$
$\vec{a}_3$	$\vec{a}_3 = [\mathcal{C}^A_{MPick}, \mathcal{C}^A_{MPlace}], \vec{x}_3 = [(a_0, b_0, \textbf{None}), (a_g, b_0, A), (a_1, b_0, \textbf{None})], \vec{u}_3 = [m_1, m_2]$

Table 2. Example walkthrough of the focused algorithm. Each $\mathcal{E}_i$ displays the set of lazy elements at the start of the iteration. Each S_i displays the set of sampler instances for which SAMPLE is called along with the new elements produced.

S_0	INITIAL-ELEMENTS$(\mathcal{P})$: $\{Stable(a_0), Stable(b_0)\}$
$\mathcal{E}_1$	$Grasp(l^G_A), Grasp(l^G_B), Stable(l^P_A), Region(l^P_A), Stable(l^P_B), Manip(a_0, l^G_A, l^A_m), Manip(b_0, l^G_B, l^B_m),$ $Manip(l^P_A, l^G_A, l^A_m), Manip(l^P_B, l^G_B, l^B_m), CFree(l^A_m, b_0), CFree(l^A_m, l^P_B), CFree(l^B_m, a_0), CFree(l^B_m, l^P_A)$
$\vec{\pi}_1$	$\vec{a}_1 = [\mathcal{C}^A_{MPick}, \mathcal{C}^A_{MPlace}], \vec{x}_1 = [(a_0, b_0, \textbf{None}), (l^G_A, b_0, A), (l^P_A, b_0, \textbf{None})], \vec{u}_1 = [l^A_m, l^A_m]$
S_1	$\psi^A_G() : Grasp(a_g), \psi^A_P() : Stable(a_1), Region(a_1)$
$\mathcal{E}_2$	$Grasp(l^G_B), Stable(l^P_B), Manip(a_0, a_g, l^A_m), Manip(a_1, a_g, l^A_m), CFree(l^A_m, b_0), CFree(l^A_m, l^P_B), CFree(l^B_m, a_0),$ $CFree(l^B_m, a_1), Manip(b_0, l^G_B, l^B_m), Manip(l^P_B, l^G_B, l^B_m)$
$\vec{\pi}_2$	$\vec{a}_2 = [\mathcal{C}^A_{MPick}, \mathcal{C}^A_{MPlace}], \vec{x}_2 = [(a_0, b_0, \textbf{None}), (a_g, b_0, A), (a_1, b_0, \textbf{None})], \vec{u}_2 = [l^A_m, l^A_m]$
S_2	$\psi^A_M(a_0, a_g) : Manip(a_0, a_g, m_1), \psi^A_M(a_1, a_g) : Manip(a_1, a_g, m_2), CFree(m_1, p_0), CFree(m_2, p_0)$
$\mathcal{E}_3$	$Grasp(l^G_B), Stable(l^P_B), CFree(m_1, l^P_B), CFree(m_2, l^P_B), Manip(b_0, l^G_B, l^B_m)$ $Manip(l^P_B, l^G_B, l^B_m), CFree(l^B_m, a_0), CFree(l^B_m, a_1)$
$\vec{a}_3$	$\vec{a}_3 = [\mathcal{C}^A_{MPick}, \mathcal{C}^A_{MPlace}], \vec{x}_3 = [(a_0, b_0, \textbf{None}), (a_g, b_0, A), (a_1, b_0, \textbf{None})], \vec{u}_3 = [m_1, m_2]$

iteration, a plan π_3 not requiring any lazy samples is generated, resulting in a solution.

The focused algorithm samples fewer values than incremental by only sampling values determined to be useful for completing lazy samples and satisfying constraints along a plan. More specifically, it avoids sampling values for object B altogether, saving time by not computing expensive motion plans. This behavior becomes even more prevalent

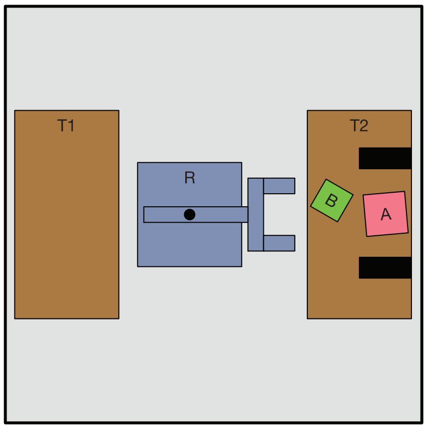


Fig. 17. Mobile manipulation problem where object B is obstructing manipulations that pick object A.

in problems with many moveable objects, such as those arising from human environments.

10.3. Additional example scenarios

We sketch out several additional problems and outline how, in particular, the focused algorithm will proceed in each of these scenarios.

10.3.1. Sampling failure. We assumed previously that each sampler successfully generated output values satisfying its constraints. In general, samplers may fail to do so because of timeouts or even because no sample exists. For example, suppose $\psi_M^A(a_0, a_g)$ fails to produce a collision-free inverse kinematic solution, resulting in a failure. After the failure, $\psi_M^A(a_0, a_g)$ will be added to *sampled*, preventing it from being sampled on the next iteration. Without $Manip(a_0, a_g, m_1)$ or $Manip(a_0, a_g, l_m^A)$, the focused algorithm will fail to find a plan. In that case, *sampled* is reset allowing $\psi_M^A(a_0, a_g)$ to be sampled again on the next episode. This cycle will automatically repeat with increased timeouts for as long as $\psi_M^A(a_0, a_g)$ fails to produce a manipulation, as picking object A is required for any solution to this problem.

10.3.2. Obstructions. Suppose that object A is initially obstructed by object B as in Figure 17. While $\psi_M^A(a_0, a_g)$ can produce a manipulation m_1, it cannot be performed because it violates the collision constraint $CFree(m_1, b_0)$. However, the lazy pose l_B^P is optimistically assumed to not be in collision with m_1. Thus, a valid plan involves first moving B to l_B^P before picking A. In the event where the sampled value for l_B^P is still in collision with m_1, an additional value can be generated on the next episode. Lazy samples allow the focused algorithm to reason about trajectories u_m that do not yet correspond to a feasible transition because they violate one or more collision constraints. By sampling concrete values for the lazy samples corresponding to these violated constraints, it can attempt to find transitions for which the control is feasible.

10.3.3. Regrasp. Consider the regrasp experiment in Figure 18 where the robot is unable to pick and place the goal object using the same grasp. Owing to this, it is forced to place the goal object at an intermediate location to change grasps. The focused algorithm will only create one lazy grasp sample l_A^G for object A. On the subsequent iteration, at least one of $\psi_M^A(a_0, a_g)$ and $\psi_M^A(a_*, a_g)$ will fail to sample a manipulation. Both will be added to *sampled* causing the discrete-search to fail to find a plan on the next iteration. After *sampled* is reset, the focused algorithm is able to use l_A^G to produce the second grasp and arrive at a solution to the problem.

11. Experiments

We implemented both algorithms and tested them on a suite of tabletop manipulation problems. All experiments used the same core factored transition system and same set of conditional samplers as described in Section 9. We wrote our conditional samplers in Python, building on top of the OpenRAVE robotics development environment (Diankov and Kuffner, 2008). We used the Open Dynamics Engine (Smith, 2005) for collision checking.

Each movable object was limited to four side grasps except for in *Experiment 1* where each object has a single-top grasp. Thus, the grasp conditional sampler ψ_G simply enumerates this finite set. The side-grasp restriction increases the difficulty of our benchmarks as it creates more opportunities for objects to obstruct each other.

Our placement conditional sampler ψ_P randomly samples poses from a mixture distribution composed of a uniform distribution over stable placements and a uniform distribution over stable placements not in collision given the initial state. This strong bias towards initially collision-free placements accelerates the generation of unobstructed placements, particularly in problems where there are many movable objects.

Our manipulation conditional sampler ψ_M samples base poses from a precomputed distribution of 2D base poses, each relative to a 2D object pose, that for some previous query admitted a kinematic solution. This "learned' base pose sampler has a greater likelihood of producing base poses that admit kinematic solutions than a sampler that generates base poses uniformly at random in a region near the desired end-effector pose. We use IKFast (Diankov, 2010) for inverse kinematics. Finally, we implemented ψ_M's sampling-based motion planner using RRT-Connect (bidirectional rapidly-exploring randomized trees) (Kuffner and LaValle, 2000).

We considered two FastDownward (Helmert, 2006) configurations for the INCREMENTAL and FOCUSED algorithms: H

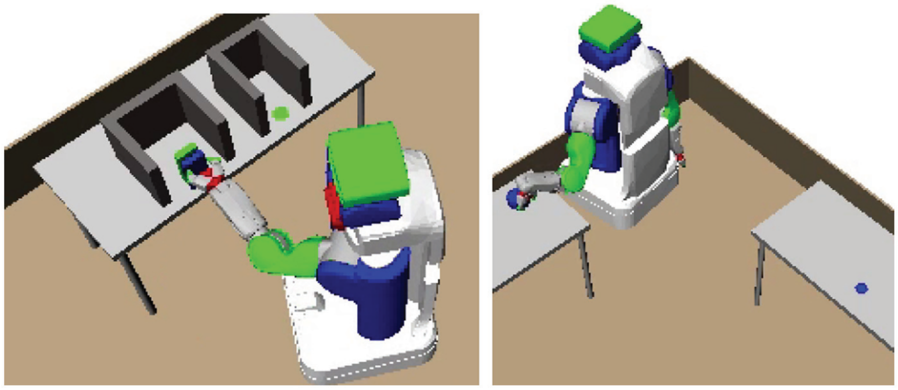


Fig. 18. *Regrasp* (left): a forced regrasp problem. *Push* (right): a problem requiring pushing, picking, and placing.

uses the FastForward heuristic (Hoffmann and Nebel, 2001) in a lazy greedy search and *No-H* is a BFS.

All trials were run on 2.8 GHz Intel Core i7 processor with a 120 second time limit. Our Python implementation of the INCREMENTAL and FOCUSED algorithms can be found at: https://github.com/caelan/factored-transition-systems. We also have developed a similar suite of algorithms for an extension of the PDDL (McDermott et al., 1998) called STRIPStream (Garrett et al., 2017b) available at https://github.com/caelan/stripstream. Videos of the experiments are available at https://youtu.be/xJ3OeMAYmgc. Base trajectories are post-processed by computing a direct trajectory between pairs of base configurations.

11.1 Scaling experiments

We performed three scaling experiments on pick-and-place problems. All experiments considered five problem sizes, varying the number of objects. We performed five trials using randomly (with the exception of *Experiment 2*) generated problem instances for each problem size. Each scatter plot in Figures 19, 20, and 21 display the total runtime of each configuration per trial. Timeouts are indicated by the omission of a trial.

Experiment 1 in Figure 22 is the "grid@tabletop" benchmark (Krontiris and Bekris, 2015) where each object has a specified goal pose. The initial placements are randomly generated. The table size scales with the number of objects. As shown in Figure 19, *Focused-H* solved all problem instances and *Incremental-H* solved all but one (size = 14) indicating that use of a heuristic is necessary for problems with long horizons.

Experiment 2 in Figure 23 has the goal that a single green object be placed in the green region. The green object is obstructed by four red objects. The number of distracting red objects on the right table is varied between 0 and 40. This experiment reflects many real-world environments where the state space is enormous but many objects do not substantially affect a task. As can be seen in Figure 20, both *Focused-No-H* and *Focused-H* solved all problem instances

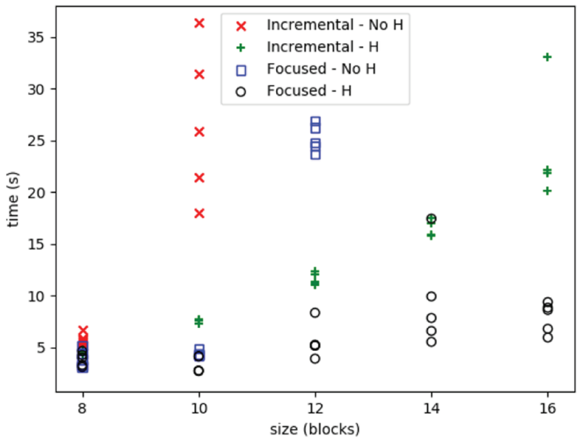


Fig. 19. *Experiment 1*: total runtime of the algorithms over five trials per problem size.

showing that the FOCUSED algorithm is able to avoid producing samples for objects until they are relevant to the task.

Experiment 3 in Figure 22 has the goal that a single blue object be moved to a different table. The blue object starts at the center of the visible table, and the red objects are randomly placed on the table. The table size scales with the number of objects. As shown in Figure 21, *Focused-H* solved all instances and *Focused-No-H* solved all but one (size = 21).

11.2. Diverse experiments

We experimented on several additional problems to show that the factored transition system framework and algorithms can be successfully applied to problems involving pushing, stacking, and discrete state variables. We also experimented on two tricky pick-and-place problems that require regrasping and require violating several goal constraints along a plan to achieve the goal. We conducted 40 trials per problem and algorithm, and each trial once again had a 120 second time limit. The success percentage of each algorithm (%) and mean runtime in seconds for successful trials are displayed in Table 3. We also show the

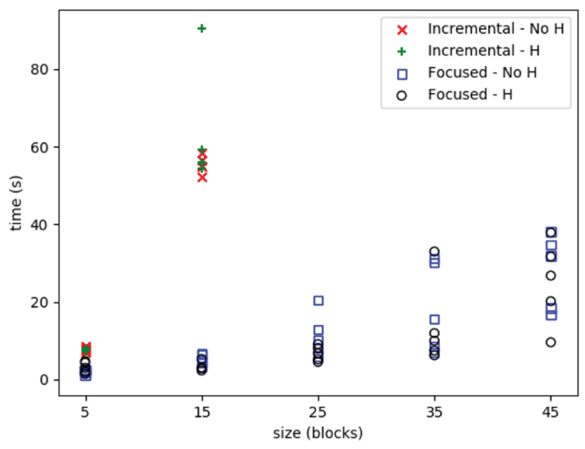


Fig. 20. *Experiment 2*: total runtime of the algorithms over five trials per problem size.

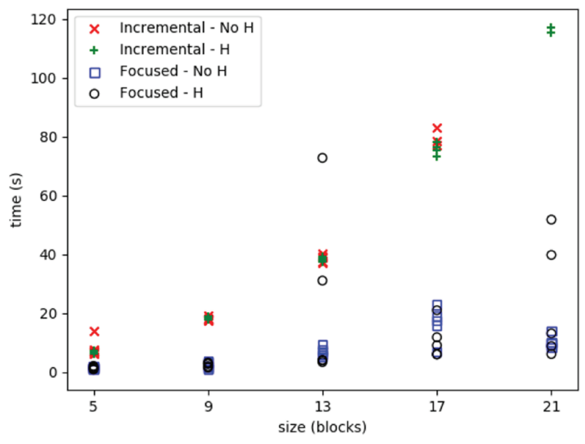


Fig. 21. *Experiment 3*: total runtime of the algorithms over five trials per problem size.

reported statistics for the best configuration of the HBF (Garrett et al., 2015) and FFRob (Garrett et al., 2017a) algorithms when applicable.

11.2.1. Descriptions. The regrasp problem (*Regrasp*) in Figure 18 is Problem 3 of Garrett et al. (2015). The goal constraints are that the green object be at the green pose and the blue object remain at its current pose. The robot must place the green object at an intermediate pose to obtain a new grasp to insert it in the thin, right cupboard. This indicates that the algorithms can solve pick-and-place problems where even non-collision constraints affect the plan skeleton of solutions.

The first pushing problem (*Push*) in Figure 18 has the goal constraint that the short blue cylinder on the left table be placed at the blue point on the right table. Because the blue cylinder is short and wide, the robot is unable to grasp it except by side grasps at the edges of each table. Thus, the robot must first push the cylinder to the edge of the left table, pick the cylinder, place the cylinder on the edge of the right table, and push the cylinder to the goal point. This problem introduces an additional transition relating to trajectories corresponding movements between two poses. In addition, it requires a conditional sampler to generate push Cartesian trajectories and motion plans per pairs of poses on the same table.

The second pushing problem (*Wall*) in Figure 24 is Problem 2 of Garrett et al. (2015) where the goal constraint is that the short green cylinder be placed at the green point. A wall of moveable objects initially blocks the robot from pushing the green cylinder to the goal point. However, if several of these blocks are moved, the robot can execute a sequence of pushes to push the green cylinder directly to its goal.

The stacking problem (*Stacking*) in Figure 25 is Problem 4 of Garrett et al. (2015). The goal constraints are that the

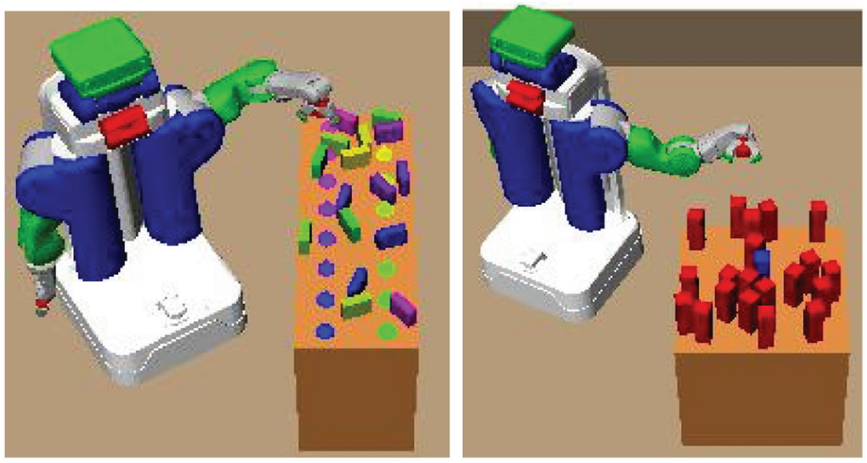


Fig. 22. *Experiment 1* (left): the robot must place each block at its corresponding goal pose. *Experiment 3* (right): the robot must move the blue block to another table.

Table 3. The success percentage (%) and mean runtime in seconds (*t*) for the diverse experiments over 40 trials compared with the reported results for the HBF (Garrett et al., 2015) and FFRob (Garrett et al., 2017a) algorithms. A dash (—) indicates that no data was available.

Problem	HBF (2015)		FFRob (2017)		*Incr.*		*Incr. - H*		*Focus*		*Focus - H*	
	%	*t*	%	*t*	%	*t*	%	*t*	%	*t*	%	*t*
Regrasp	100	6	—	—	98	1	100	2	98	1	95	1
Push	—	—	—	—	100	11	100	13	100	13	100	9
Wall	100	7	—	—	95	10	98	13	100	6	100	8
Stacking	97	12	—	—	100	9	100	9	100	2	100	3
Non-mon.	—	—	72	135	25	21	98	15	0	—	88	43
Dinner	—	—	74	44	0	—	100	27	0	—	98	22

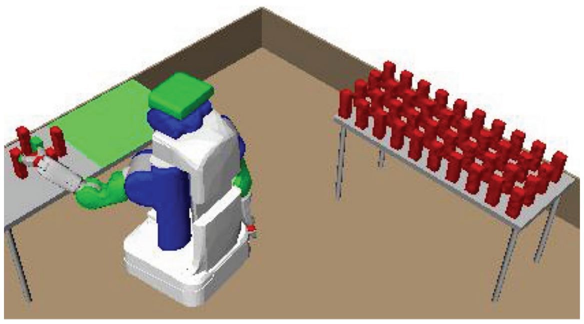


Fig. 23. *Experiment 2*: the robot must place the green object in the green region.

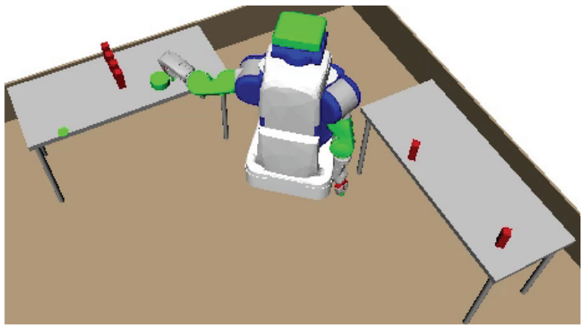


Fig. 24. *Wall*: a pushing problem involving a wall of blocks.

blue block be contained within the blue region, the green block be contained within the green region, and the black block be on top of the blue block. The robot must unstack the red block to safely move the green block. This problem requires a modification of the transition system to account for stability constraints. In addition, it requires a conditional sampler that produces poses of the black block on the blue block given poses of the blue block.

The pick-and-place problem (*Non-mon.*) in Figure 26 is Problem 3-2 of Garrett et al. (2017a). The goal constraints are that the green blocks be moved from their initial pose on the left table to their corresponding pose on the right table. In addition, there are goal constraints that each blue and cyan block end at its initial pose. This is a highly non-monotonic problem as solutions require violating several goal constraints satisfied by the initial state.

The task and motion planning problem (*Dinner*) in Figure 27 is Problem 5 of Garrett et al. (2017a). The state contains an additional discrete state variable for each block indicating whether it is *dirty, clean*, or *cooked*. The transition relation contains additional clauses to clean a dirty block when it is placed on the dishwasher and to cook a clean block when it is placed on the microwave. The goal constraints are that the green blocks (“cabbage”) be cooked and placed on the plates, the blue blocks (“cups”) be cleaned and placed at the blue points, the cyan block (an unnecessary

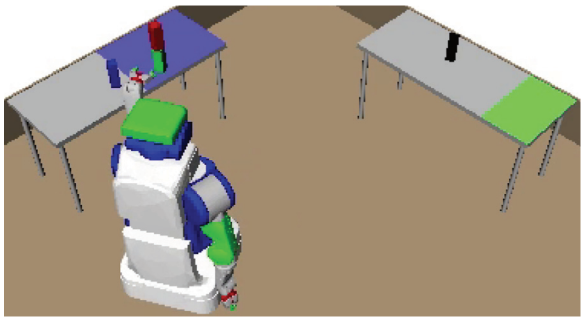


Fig. 25. *Stacking*: a problem requiring unstacking and stacking.

“cup”) be cleaned, and the pink blocks (“turnips”) remain placed on the shelf. To reach the cabbage, the robot must first move several turnips and the later replace them to keep the kitchen tidy.

11.2.2. Results. Each algorithm performed comparably on the first four problems (*Regrasp, Push, Wall*, and *Stacking*). When compared with HBF (Garrett et al., 2015), *Focused-H* has a slightly lower average runtime for *Regrasp* and *Stacking* and about the same average runtime for *Wall*. However, the average runtime for all algorithms on these problems is less than 15 seconds. Only the heuristically

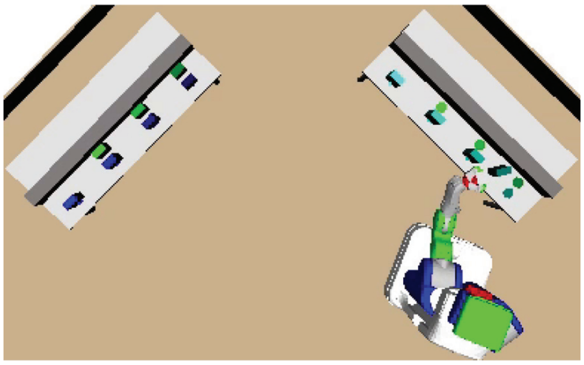


Fig. 26. *Non-mon.*: a non-monotonic pick-and-place problem.

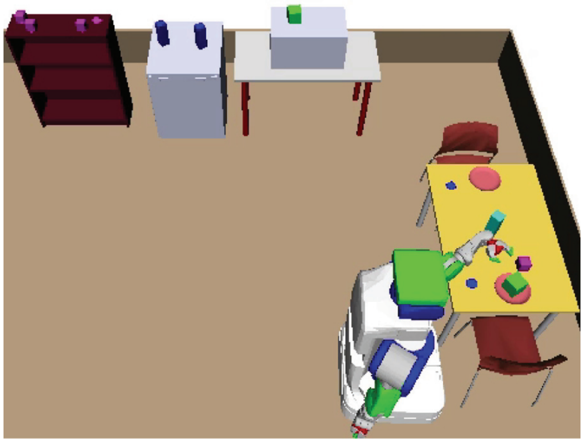


Fig. 27. *Dinner*: a task and motion planning problem.

informed algorithms where able to consistently solve the larger last two problems (*Non-mon., Dinner*). For problem *Non-mon., Incremental-H* slightly outperformed *Focused-H* because this problem requires manipulating each object. Thus, *Incremental-H* and *Focused-H* produce comparable sets of samples, but *Incremental-H* has less overhead. Both algorithms performed significantly better than best algorithm of Garrett et al. (2017a). For problem *Dinner*, the heuristic guided *Incremental-H* and *Focused-H* planners were able to quickly solve the problem reinforcing the point that search guidance is necessary for problems over long horizons. These algorithms compare favorably to the best algorithm of Garrett et al. (2017a).

12. Conclusion

We introduced factored transition systems for specifying planning problems in discrete-time hybrid systems. Factored transition systems can model motion planning, pick-and-place planning, and task and motion planning applications. The transition dynamics for multi-object manipulation are significantly factorable. Legal transitions can be expressed as the conjunction of constraints each involving only several state or control variables.

Conditional constraint manifolds enabled us to give a general definition of robust feasibility for factored transition systems. Under certain conditions, they allow us to describe a submanifold of plan parameter-space resulting from the intersection of dimensionality-reducing constraints. Thus, robustness properties can be examined relative to this submanifold rather than for the plan parameter space. We introduced the idea of conditional samplers: samplers that given input values, produce a sequence of output values satisfying a constraint with the input values. When appropriate conditional samplers are specified for each conditional constraint manifold, the resulting collection of samplers is sufficient for solving any robustly feasible problem. Sampling benefits from factoring because a small collection of samples for each variable can correspond to a large number of combined states and transitions.

We gave two general-purpose algorithms that are probabilistically complete given sufficient samplers. The incremental algorithm iteratively calls each conditional sampler and tests whether the set of samples is sufficient using a blackbox, discrete search. The focused algorithm first creates lazy samples representing hypothetical outputs of conditional samplers and then uses a blackbox, discrete search to identify which lazy samples could be useful. We empirically demonstrated that these algorithms are effective at solving challenging pick-and-place, pushing, and task and motion planning problems. The focused algorithm is more effective than the incremental algorithm in problems with many movable objects as it can selectively produce samples for only objects affecting the feasibility of a solution. In addition, both algorithms were more efficient when using a discrete search subroutine that exploited factoring in the search through domain-independent heuristics.

12.1. Future work

Future work will involve developing additional algorithms for solving factored transition systems. In particular, both the incremental and focused algorithms treat discrete-search as a blackbox. By directly integrating the search and sampling, an algorithm may be able to more directly target sampling based on the search state and possible transitions. For example, backward-forward search (Garrett et al., 2015) performs its search directly in the hybrid state space instead of a discretized state space.

Our formulation gives rise to several new opportunities for learning to improve sampling and search runtimes. For instance, one could learn a policy to decide how frequently to sample each conditional sampler. A high-performing policy would balance the likelihood of a conditional sampler to produce useful samples, the overhead of computing samples, and the impact additional samples have on subsequent discrete searches. Similarly, in the focused algorithm, one could learn costs associated with using lazy samples reflective of the expected future planning time resulting from sampling from a particular conditional stream. These costs could cause discrete-search to produce plans that are likely realizable without too much overhead.

Finally, this work can be extended to optimal planning settings where there are non-negative costs $c(\bar{u})$ on control inputs. This will require adapting properties such as asymptotic optimality (Karaman and Frazzoli, 2011) to the factored transition system setting and modifying the incremental and focused algorithms to achieve these properties.

Acknowledgments

We thank Beomjoon Kim, Ferran Alet, and Zi Wang for their feedback on this manuscript.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: We gratefully acknowledge support from the NSF (grant numbers 1420316, 1523767, and 1723381), from the AFOSR (grant number FA9550-17-1-0165), and from the ONR (grant number N00014-14-1-0486). Caelan Garrett is supported by an NSF GRFP fellowship (primary award number 1122374). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

Alami R, Laumond JP and Siméon T (1994) Two manipulation planning algorithms. In: *Workshop on Algorithmic Foundations of Robotics (WAFR)*.

Alami R, Siméon T and Laumond JP (1990) A geometrical approach to planning manipulation Tasks. the case of discrete placements and grasps. In: *International Symposium of Robotic Research (ISRR)*.

Alur R, Courcoubetis C, Halbwachs N, et al. (1995) The algorithmic analysis of hybrid systems. *Theoretical Computer Science* 138(1): 3–34.

Alur R, Henzinger TA, Lafferriere G and Pappas GJ (2000) Discrete abstractions of hybrid systems. *Proceedings of the IEEE* 88(7): 971–984.

Bäckström C and Nebel B (1995) Complexity results for SAS + planning. *Computational Intelligence* 11(4): 625–655.

Barraquand J, Kavraki L, Latombe JC, Motwani R, Li TY and Raghavan P (1997) A random sampling scheme for path planning. *The International Journal of Robotics Research* 16(6): 759–774.

Barry J, Kaelbling LP and Lozano-Pérez T (2013) A hierarchical approach to manipulation with diverse actions. In: *2013 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1799–1806.

Barry JL (2013) *Manipulation with Diverse Actions*. PhD Thesis, Massachusetts Institute of Technology.

Berenson D and Srinivasa SS (2010) Probabilistically complete planning with end-effector pose constraints. In: *2010 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2724–2730.

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

Bonet B and Geffner H (2001) Planning as heuristic search. *Artificial Intelligence* 129(1): 5–33.

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

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

de Silva L, Pandey AK, Gharbi M and Alami R (2013) Towards combining HTN planning and geometric task planning. In: *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

Dechter R (1992) Constraint networks. Technical Report, Information and Computer Science, University of California, Irvine. Available at: http://www.ics.uci.edu/~csp/r17-survey.pdf (accessed 9 September 2018).

Dellin CM and Srinivasa SS (2016) A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. In: *International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Diankov R (2010) *Automated Construction of Robotic Manipulation Programs*. PhD Thesis, Robotics Institute, Carnegie Mellon University.

Diankov R and Kuffner J (2008) Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Carnegie Mellon University.

Dornhege C, Eyerich P, Keller T, Trüg S, Brenner M and Nebel B (2009) Semantic attachments for domain-independent planning systems. In: *International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, pp. 114–121.

Dornhege C, Hertle A and Nebel B (2013) Lazy evaluation and subsumption caching for search-based integrated task and motion planning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on AI-based robotics*.

Edelkamp S (2004) Pddl2.2: The language for the classical part of the 4th international planning competition. In: *4th International Planning Competition (IPC'04), at ICAPS'04*.

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

Erol K, Hendler J and Nau DS (1994) HTN planning: Complexity and expressivity. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence*, vol. 2. AAAI Press, pp. 1123–1128.

Fikes RE and Nilsson NJ (1971) STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2: 189–208.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2014) FFRob: An efficient heuristic for task and motion planning. In: *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2015) Backward–forward search for manipulation planning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Garrett CR, Lozano-Perez T and Kaelbling LP (2017a) FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research* 37(1): 104–136.

Garrett CR, Lozano-Pérez T and Kaelbling LP (2017b) Strips planning in infinite domains. Preprint arXiv:1701.00287.

Hauser K and Latombe JC (2010) Multi-modal motion planning in non-expansive spaces. *The International Journal of Robotics Research* 29: 897–915.

Hauser K and Ng-Thow-Hing V (2011) Randomized multi-modal motion planning for a humanoid robot manipulation task. *The International Journal of Robotics Research* 30(6): 676–698.

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

Henzinger TA (2000) The theory of hybrid automata. In: *Verification of Digital and Hybrid Systems*. Berlin: Springer, pp. 265–292.

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

Hsu D, Latombe JC and Motwani R (1997) Path planning in expansive configuration spaces. In: *Proceedings 1997 IEEE International Conference on Robotics and Automation*, vol. 3. IEEE, pp. 2719–2726.

Jensen FV (1996) *An Introduction to Bayesian Networks*, vol. 210. London: UCL Press.

Kaelbling LP and Lozano-Pérez T (2011) Hierarchical planning in the now. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

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

Kavraki LE, Kolountzakis MN and Latombe JC (1998) Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation* 14(1): 166–171.

Kavraki LE and Latombe JC (1998) Probabilistic roadmaps for robot path planning. *Practical Motion Planning in Robotics: Current Approaches and Future Directions*. New York: John Wiley & Sons, Inc.

Kavraki LE, Latombe JC, Motwani R and Raghavan P (1995) Randomized query processing in robot path planning. In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*. New York: ACM Press, pp. 353–362.

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

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

Krontiris A and Bekris KE (2016) Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner. In: *International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden.

Kuffner JJ Jr and LaValle SM (2000) RRT-Connect: An efficient approach to single-query path planning. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

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

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

Laumond JPP (1998) *Robot Motion Planning and Control*. Secaucus, NJ: Springer-Verlag New York, Inc.

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

Lozano-Pérez T (1981) Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man, and Cybernetics* 11: 681–698.

Lozano-Pérez T, Jones JL, Mazer E, Lanusse A, et al (1987) Handey: A robot system that recognizes, plans, and manipulates. In: *IEEE International Conference on Robotics and Automation (ICRA)*.

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

McDermott D, Ghallab M, Howe A, et al. (1998) PDDL: The planning domain definition language. Technical Report, Yale Center for Computational Vision and Control.

Pandey AK, Saut JP, Sidobre D and Alami R (2012) Towards planning human–robot interactive manipulation tasks: Task dependent and human oriented autonomous selection of grasp and placement. In: *RAS/EMBS International Conference on Biomedical Robotics and Biomechatronics*.

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

Siméon T, Laumond JP, Cortés J and Sahbani A (2004) Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research* 23(7–8): 729–746.

Smith R (2005) ODE: Open Dynamics Engine User Guide. Available at: http://ode.org/ode-latest-userguide.pdf

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

Stilman M and Kuffner JJ (2006) Planning among movable obstacles with artificial constraints. In: *Workshop on Algorithmic Foundations of Robotics (WAFR)*.

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

Thiébaux S, Hoffmann J and Nebel B (2005) In defense of PDDL axioms. *Artificial Intelligence* 168(1–2): 38–69.

Torrisi FD and Bemporad A (2001) Discrete-time hybrid modeling and verification. In: *Proceedings if the 40th IEEE Conference on Decision and Control*, Vol. 3. IEEE, pp. 2899–2904.

Toussaint M (2015) Logic-geometric programming: an optimization-based approach to combined task and motion planning. In: *AAAI Conference on Artificial Intelligence*. AAAI Press, pp. 1930–1936.

Toussaint M and Lopes M (2017) Multi-bound tree search for logic-geometric programming in cooperative manipulation domains. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4044–4051.

Tu LW (2010) *An Introduction to Manifolds*. Berlin: Springer.

Van Den Berg J, Stilman M, Kuffner J, Lin M and Manocha D (2009) Path planning among movable obstacles: A probabilistically complete approach. In: *Algorithmic Foundation of Robotics VIII*. Berlin: Springer, pp. 599–614.

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

Vendittelli M, Laumond JP and Mishra B (2015) Decidability of robot manipulation planning: Three disks in the plane. In: *Algorithmic Foundations of Robotics XI*. Berlin: Springer, pp. 641–657.

Wilfong GT (1988) Motion planning in the presence of movable obstacles. In: *Symposium on Computational Geometry*. Berlin: Springer, pp. 279–288.