A constraint-based method for solving sequential manipulation planning problems

Tomás Lozano-Pérez and Leslie Pack Kaelbling

Abstract— **In this paper, we describe a strategy for integrated task and motion planning based on performing a symbolic search for a sequence of high-level operations, such as pick, move and place, while postponing geometric decisions. Partial plans (skeletons) in this search thus pose a geometric constraint-satisfaction problem (CSP), involving sequences of placements and paths for the robot, and grasps and locations of objects. We propose a formulation for these problems in a discretized configuration space for the robot. The resulting problems can be solved using existing methods for discrete CSP.**

I. INTRODUCTION

Much of the work on robot motion planning has focused on planning a single path with a specified starting and ending configuration. It has long been recognized [1], [2], [3], [4], [5], [6], [7], however, that solving manipulation problems requires planning a coordinated sequence of motions that involve picking, placing, pushing, or otherwise manipulating the objects, as well as moving through free space. However, the dimensionality (and therefore the running time) of these manipulation planning problems increases substantially over traditional single motion-planning problems.

This increased dimensionality motivates a hierarchical *task and motion planning* (TAMP) approach, in which a symbolic search at a more abstract level of representation is used to guide the search for a sequence of motion plans that achieve a desired objective in the environment (for example, that some objects are placed in some region of space).

There has been increasing interest in the TAMP problem, with many suggested solutions for addressing the difficult multi-level search problem it entails [8], [9], [10], [11], [12], [13], [14]. Most methods ultimately involve a search at the task level in an abstract space, in which determining whether an operation is legal depends on the solution of a high-dimensional geometric motion-planning problem. Such approaches are particularly difficult to manage when a geometric decision made early in the high-level plan affects the feasibility of the rest of the plan in a way that is not revealed until later in the search. Dependencies of this kind can lead to relatively unguided backtracking.

An alternative approach [13], [15] is for the task-level planner to delay the binding of any geometric choices (grasps, object placements, robot configurations, paths) until it has constructed a plan "skeleton" that consists of unparameterized robot operations, such as *pick b*, *move* and *place b*, together with a set of constraints on the details of those operations. Then, a geometric planner can attempt to "fill in the details." The advantage of this strategy is that the interrelated geometric decisions can be considered collectively, allowing us to bring insights from constraint-satisfaction solvers to bear on the problem. For example, the most constrained decisions can be addressed first, independent of whether they come early or late in the plan.

Note that in these constraint-based approaches, the geometric planner is in the inner loop of a symbolic planner, and must quickly decide whether a set of constraints is satisfiable. However, the constraints generally describe complicated non-convex regions in some underlying parameter space. The solution we describe deterministically finds solutions for constraints in a quantized representation of the task parameters. The fidelity of the representation can be increased with a corresponding increase in computational cost. The use of a discrete representation allows us to use solution methods from the CSP (Constraint Satisfaction Problem) literature [16].

This basic approach is not specific to any particular robot, primitives, or tasks. In this paper, we describe a formulation of plan skeletons for a mobile manipulation problem with pick-and-place operations, show how they yield constraint satisfaction problems, and how these problems are ultimately solved. We demonstrate the flexibility of the method by showing that the same basic skeleton with slightly different constraints can give rise to solutions to multiple problems, including moving one object out of the way in order to grasp another, putting two objects into the same constrained region of space, and regrasping an object so that it can be placed into a tight space. We explore the performance of several variations on the search strategy for constraint satisfaction and on the method used for motion planning, and document how the method scales with sampling granularity and problem complexity.

II. RELATED WORK

Our focus is on manipulation problems. Early work in this area [1] used somewhat ad-hoc techniques to couple planning choices; for example, allowing the choice of a grasp to depend on the choice of placement location. The Handey system [2], [17] extended these techniques, using an explicit graph search to find a sequence of regrasping motions.

In their seminal work, Alami et al. [18] and Simeon et al. [4] defined the *manipulation graph* of "transfer" and

This work was supported in part by the NSF under Grant No. 019868, in part by ONR MURI grant N00014-09-1-1051, in part by AFOSR grant AOARD-104135 and in part by Singapore Ministry of Education under a grant to the Singapore-MIT International Design Center.

Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA tlp@mit.edu, lpk@mit.edu

"transit" motions, in which the search for grasping, placement and movement actions are integrated. Their approach involves building probabilistic road-maps embedded in the appropriately constrained spaces. Hauser [5] generalized these ideas to a broader class of tasks (such as walking and pushing) using the idea of *multi-modal* planning, where a mode roughly corresponds to a constrained subspace in the task parameter space. Since sample-based planners do not naturally terminate when solutions do not exist, Hauser developed a search strategy that allocates search effort among multiple possible mode transitions. This line of work is directly relevant to our problem, although it does not address all of the constraints, especially those involving manipulating a sequence of different objects, required to instantiate the plan skeletons we consider. Also, we explore a strategy based on deterministic (resolution limited) sampling which can determine infeasibility (at a given resolution).

More recent work in TAMP has addressed similar problems. The most relevant work is that of Lagriffoul et al. [13], who develop a constraint-satisfaction approach to a subset of the geometric problems arising in TAMP. They do a backtracking search over a discrete set of possible object grasps and placements; but given a partial assignment of those values, they are able to formulate a set of linear constraints on robot configurations and object poses that allow them to efficiently determine which assignments are feasible and rule out large numbers of useless branches, significantly limiting backtracking. For each step in the plan skeleton, they call an RRT to determine feasibility of the motion. Their approach can be thought of, in terms of constraint satisfaction approaches, as doing backtracking with forward checking [20] of the kinematic constraints (but, importantly, not the path-existence constraints) and a fixed variable ordering. Our approach, in contrast, treats path existence as another constraint, which can also be used in forward checking; it also can choose to consider the most constrained decision first, independent of where it occurs in the plan skeleton. Perhaps because of this, we are able to operate with considerably larger numbers of possible discrete placements and grasps.

Pandey et al. [21] describe a system that chooses geometric parameters for tasks involving both a robot and a human, reasoning about what is feasible for the robot and the human, and taking into account the level of effort required from each participant by each of the choices. The geometric choices are made by filtering out locally inconsistent values and backtracking over the remaining values.

III. TASK-LEVEL PLANNING

We use a simple task-level planner, in which operators are described with two types of preconditions: symbolic and geometric. Symbolic preconditions are described using symbolic fluents as is typical in symbolic planning formalisms. Geometric preconditions are described using constraints on a set of constraint variables that remain unbound during the process of task-level planning. A *constraint variable* is a variable that ranges over a continuous geometric quantity such as an object pose or robot configuration.

Associated with each planning operator is a *primitive* action that can be taken by the robot; it is typically specified with symbols naming objects and with a set of geometric parameters. A primitive will, in general, correspond to a closed-loop robot control program.

Whenever the symbolic search process reaches a state in which the symbolic aspects of the goal is satisfied, we extract a *plan skeleton*, which is a sequence of primitives that have constant values for the symbolic parameters, but whose geometric parameters are instantiated with constraint variables. A plan skeleton is accompanied by a set of constraints that relate the constraint variables that occur in the primitives to one another as well as to quantities in the goal specification of the plan and in the initial state description.

We call the CSP solver to see if the constraints associated with the plan skeleton are satisfiable. If so, the resulting variable bindings are used to bind the constraint variables in the plan. If not, then the planner continues to search.

IV. PROBLEM FORMULATION

The mechanism for symbolic planning with deferred constraint satisfaction is general, and we have a domain-independent implementation of it. In this paper, we describe a particular application of this method to solving pick-and-place problems for a one-armed mobile manipulation robot. Our formulation of this domain has three primitives: $pick$, $move$, and $place$. $Pick$ and $place$ are assumed to be very simple parameterless policies that either move forward and adaptively close the hand or open the hand and retract. The correctness of a plan skeleton is enforced by the constraints, which are expressed as relationships between poses of objects in the world, poses of the robot hand, and configurations of the rest of the robot, as well as requirements that necessary clear paths exist.

To formulate a domain as a discrete constraint satisfaction problem (CSP), it is necessary to specify: a set of constraint variables, a discrete domain of values for each variable, and a set of constraints. Constraints are specified by a set of variables to which they apply and an arbitrary test function that maps an assignment of variable values to $True$ or $False$.

A. Constraint variables

We will use a *hand-centric* representation of the robot's configurations and relations to other objects, which allows the constraint satisfaction problem to be described more compactly and solved more efficiently. It is encoded using variables of the following three types:

- H: Cartesian poses of the robot's hand or end effector. In general, the domain of H is the set of six-degree-of-freedom poses within the workspace of the mobile manipulator.
- K: Configurations of the robot. The domain of K is the set of robot configurations; in our domain it is the 7 degrees of freedom of the arm.

- G: Grasps, which are six-degree-of-freedom specifications of the pose of an object relative to the hand.

It is typical to think of a robot configuration being described with a Cartesian base pose and an arm configuration. The advantage of that representation is that the only constraints on validity are joint limits and collisions. The disadvantage of that representation is that the typical Cartesian motions required for manipulation lie on a complicated constraint surface in this space. In the hand-centric representation, we use the hand's Cartesian pose and the arm configuration to parameterize the configuration space (C-space). The space of valid configurations is now on a constrained surface (the robot base must be on the floor and not in collision) but the manipulation motions may be more easily described. In our current implementation, we work in a discretized subspace of this general C-space (see section VI for details).

We use some additional types of constant quantities in the specification of constraints, which are determined by the goal condition and/or initial conditions of the problem: p ranges over Cartesian poses of objects and r ranges over regions of the workspace. We will write $H \circ G$ to specify the composition of a hand pose H with a grasp G; because a grasp describes the relation between hand and object, $H \circ G$ can also be interpreted as an object pose.

B. Constraints

Constraints relate constraint variables to one another and to constant quantities; if they are collectively satisfiable, then a plan matching the skeleton exists and the satisfying bindings of the constraint variables specify the details of the geometric solution. In the pick-and-place domain, we use constraints of the following types:

- $Grasp(o, p, H, G)$: If an object o is at pose p and the hand is at pose H, then the pose of the object relative to the hand is G. This can also be written $H \circ G = p$.
- $Contained(o, p, r)$: If object o is at pose p, it will be contained in region r.
- $Disjoint(o_1, p_1, o_2, p_2)$: If object o_1 is at pose p_1 and object o_2 is at pose p_2, then the two objects are not in collision with one another.
- $\exists$VP$(H, K, o, G, \mathcal{O})$: There exists a valid path from a fixed "home" configuration to the configuration (H, K), while holding object o in grasp G. A path is valid if it avoids all fixed obstacles in the domain as well as the obstacles in set $\mathcal{O}$ and the base is on the floor in configuration (H, K) and stays on the floor throughout the path. If the robot is not holding an object during this path, then o and G will be $\emptyset$.

Because our goal is to determine feasibility of plan skeletons, and because we generally want to avoid having the robot block itself in by placing objects in such a way that it cannot return to its original configuration, we require all robot configurations to be reachable from a "home" configuration. If two configurations are reachable from home, then they will be reachable from one another. This assumption will allow us to consider $O(N)$ rather than $O(N^2)$ paths, where N is the number of configurations. But, it does prevent the technique from being appropriate for solving NAMO-type problems [22].

primitive	constraints
$move(H_1, K_1)$	$\exists$VP$(H_1, K_1, \emptyset, \emptyset, \{a@p_a\})$
$pick(a, G_1)$	$Grasp(p_a, H_1, G_1)$
$move(H_2, K_2)$	$\exists$VP$(H_1, K_1, a, G_1, \{\ \})$ $\exists$VP$(H_2, K_2, a, G_1, \{\ \})$
$place(a, G_2)$	$G_1 = G_2$ $Contained(a, H_2 \circ G_2, r)$ $\exists$VP$(H_2, K_2, \emptyset, \emptyset, \{a@H_2 \circ G_2\})$

TABLE I: Skeleton for pick and place

In the context of a planning loop, we will prize efficiency of computation over the optimality of paths, and so finding a path between two configurations that goes via home is acceptable. Once a final feasible skeleton is found, it will be possible to use this or other methods to plan more carefully and find paths that are more nearly optimal.

V. EXAMPLE PROBLEMS

To build intuition, we will illustrate the use of primitive actions and constraints to construct plan skeletons for four different tasks. The general framework of building a plan skeleton and then solving a constraint satisfaction problem can be applied to any set of action primitives; in our examples we restrict our attention to pick, move, and place operations. For brevity, these examples involve the manipulation of only one or two objects, but the skeletons can involve arbitrarily many motion and manipulation steps.

a) Basic pick and place: We will begin with a simple pick-and-place problem, in which the goal is to put an object a in some region of space r. The plan to achieve this goal has four steps: $move(H_1, K_1)$, to get the robot to a configuration from which a can be picked; $pick(a, G_1)$, to close the hand around the object with grasp G_1; $move(H_2, K_2)$ to get the robot to a configuration from which a can be placed; and $place(a, G_2)$ to release the gripper and retract the hand slightly. Table I shows the constraints associated with each step of this plan. Variables of the form p_o denote the initial pose of object o, and $o@p$ denotes an object shape o placed at pose p.

The first constraint is that there be a clear path to move, with an empty hand, to configuration H_1, K_1, while avoiding object a in its initial pose (as well as all objects that are never mentioned in this plan, in their current poses—this condition will be in force throughout all plans). The second is that, with the hand at pose H_1, it be possible to grasp object a at its initial pose using grasp G_1. The next two constraints are path constraints: they require that it be possible to move from the home configuration to H_1, K_1 and to H_2, K_2 while holding object a in grasp G_1 and not avoiding any additional obstacles. Finally, we have a place operation with grasp G_2 which must equal G_1. In addition, when object a is at pose $H_2 \circ G_2$, it must be resting stably and be contained in region R. Finally, we add a constraint that the robot be able to

primitive	constraints
$move(H_1, K_1)$	$\exists$VP$(H_1, K_1, \emptyset, \emptyset, \{a@p_a, b@p_b\})$
$pick(b, G_1)$	$Grasp(p_b, H_1, G_1)$
$move(H_2, K_2)$	$\exists$VP$(H_1, K_1, b, G_1, \{a@p_a\})$ $\exists$VP$(H_2, K_2, b, G_1, \{a@p_a\})$
$place(b, G_2)$	$G_1 = G_2$ $\exists$VP$(H_2, K_2, \emptyset, \emptyset, \{a@p_a, b@H_2 \circ G_2\})$
$move(H_3, K_3)$	$\exists$VP$(H_3, K_3, \emptyset, \emptyset, \{a@p_a, b@H_2 \circ G_2\})$
$pick(a, G_3)$	$Grasp(p_a, H_3, G_3)$
$move(H_4, K_4)$	$\exists$VP$(H_3, K_3, a, G_3, \{b@H_2 \circ G_2\})$ $\exists$VP$(H_4, K_4, a, G_3, \{b@H_2 \circ G_2\})$
$place(a, G_4)$	$G_3 = G_4$ $Contained(a, H_4 \circ G_4, R)$ $Disjoint(a, H_4 \circ G_4, b, H_2 \circ G_2)$ $\exists$VP$(H_4, K_4, \emptyset, \emptyset, \{a@H_4 \circ G_4\})$

TABLE II: Skeleton for moving an object out of the way

disengage from placing a, by requiring a clear path from home to H_2, K_2, holding nothing, but avoiding object a at its final pose.

b) Moving an object out of the way: In a situation in which the robot cannot reach object a directly, the previous set of constraints will be unsatisfiable; due to further automated high-level planning, we might develop a plan skeleton of the form shown in table II, which will allow the robot to move the obstacle b out of the way.

The skeleton resembles two copies of the simple pick-and-place plan skeleton concatenated together. The critical connection is that, rather than specifying a region in which object b should be placed, we require only that it should be placed in such a way as to allow a clear paths for picking and placing object a.

c) Putting two objects into the same constrained region: Interestingly, a very similar plan skeleton applies to a situation in which two objects must be placed into the same single region, r. The only change is the addition of a constraint on the placement of b:

$$Contained(b, H_2 \circ G_2, r) \ .$$

The requirement that it be possible to pick and place a when b is located at its new pose will constrain the choice of poses for a and b in region r so that they can be placed in order: b first, and then a.

d) Regrasping: Another situation in which the simple pick-and-place plan skeleton is not satisfiable is when there is no single grasp that is feasible at both the pick and place locations. In such cases, we must pick the object, place it in a more open region of space, then pick it again with a new grasp that will suffice for the final placement. The plan skeleton for regrasping is also a minor variation on the one for moving an object out of the way: it involves two pick-and-place sequences of the same object, but using different grasps.

VI. SOLUTION STRATEGY

The constraints from a plan skeleton are significantly non-linear and much too complex to solve exactly analytically in continuous form. We discretize the value domains of the variables and use a constraint solver for variables with discrete domains. Classical constraint solvers assume that constraints involve only two variables and that the variables have small domains. Plan skeletons have constraints with many variables and the variables have large domains. However, there has been extensive work on CSP solvers and some more modern solvers are effective for problems such as the one we face.

One key advantage of a CSP formulation is that it reduces our job to picking variables and constraints to represent the problem, and uses a generic solver to do the search. It is generally easier to articulate and check constraints for a given assignment of the variables than to construct a problem-specific search strategy. However, the constraint checks must be very efficient, since they will be called many times by the solver. The fundamental question, then, is how we should represent the variables and evaluate the constraints articulated in the previous section.

A. Variables

In this section, we develop discrete representations for H, K, and G variables. Naive discretization of the original high-dimensional spaces would result in an intractable problem, so we must reduce the domain sizes while retaining sufficient expressive power to state problems of interest.

A *hand pose*, H, is a six-degree-of-freedom Cartesian pose for the end-effector coordinate frame of the robot. We will represent it with four discrete-valued variables, H_x, H_y, H_z, H_θ, where H_θ represents rotation in the horizontal plane. The domains of these variables, $\mathcal{H}_x, \mathcal{H}_Y, \mathcal{H}_Z, \mathcal{H}_\theta$, are sets of discrete values in the Cartesian workspace and reachable horizontal rotational range of the robot. We are assuming that objects will be resting in a stable configuration on a horizontal surface when they are not in the hand, and as such have only four degrees of freedom. Note that the discrete set of grasps considered can involve arbitrary orientations of the gripper in the "hand" frame, so the 4 DOF parameterization still enables a rich class of grasps. The H_x and H_y values are uniformly spaced; the H_z values are chosen to be 'relevant' values where the C-space constraints change, and the H_θ values are non-uniformly spaced with more values near candidate grasping hand orientations for the initial poses of the objects.

A *robot kinematic configuration* for a PR2 consists of 7 degrees of freedom connecting the hand frame to the base frame. Given a fixed hand frame, we can think of this as a redundant parameterization of the base frame of the robot. However, most of the base poses are invalid: we are only interested in those that put the base on the floor. So, we will think of K, instead, as a set of functions that map a hand pose H into a valid base pose.

Let $\mathcal{B}$ be the set of valid base poses, let $\beta(H) \subset \mathcal{B}$ be the subset of $\mathcal{B}$ that is reachable with the hand fixed at H and let K_i be a function that maps hand pose H to a particular $B \in \beta(H)$. We will say that K_i is ϵ-*robust* if for any H' in an ϵ ball around H, $K_i(H') \in \beta(H')$. That is, if we apply the function to a slightly different hand pose, it will yield a base pose that is kinematically feasible for that hand pose. We are interested in robust kinematic solutions because, ultimately,

we wish to use this planner on a real robot, where uncertainty is inevitable, and it is important to have solutions that can be locally modified in response to changes in estimated object or robot poses. We perform off-line processing to find a discrete set of robust functions K_i, and it is this set of values[1] that make up $\mathcal{K}$, the domain of the variable K in our formulation.

We handle the grasp variables G similarly, drawing their values from a discrete domain $\mathcal{G}$, corresponding to a set of predetermined discrete candidate grasps for each object type; the elements of $\mathcal{G}$ actually specify a "pre-grasp" hand pose, from which the grasp primitive can be executed.

Therefore, our parameterization for a robot configuration, possibly holding an object, is specified with 6 discrete values: $(H_x, H_y, H_z, H_\theta, K, G)$. It is important to note that we do not need to check kinematic feasibility given these values; we have taken care of this in the definition of the parameters.

The constraint examples given earlier mention object poses and regions but these are not variables in the CSP formulation. It should be clear that object shapes, initial poses of objects and shapes of regions are all used in checking the constraints but are not chosen during the solution process. Even placements of objects that are chosen during the solution are not explicitly represented as such, but are a consequence of the hand and grasp parameters used for placing them.

B. Constraints

To specify a CSP, together with discrete value domains for each variable, one must specify functions that can check whether a complete or partial assignment of values to variables satisfies each constraint. For ease of understanding, in section IV-B, we described the constraints in terms of objects and poses as well as H, K, and G variables; now we must articulate them in terms only of H, K, and G, with the other quantities "built into" the constraint testing functions.

The $Disjoint$ constraint is actually subsumed by the ∃VP constraint, because the robot will not be allowed to move into a configuration that would cause two objects to collide.

The $Grasp(o, p, H, G)$ constraint can be rewritten, for a constant pose p, as $Grasp_{o,p}(H, G)$; the corresponding test on variables H and G is whether, if the robot hand is at H holding o with grasp G, then the pose of o is p. Similarly, the $Contained(o, p, r)$ constraint can be rewritten as a constraint on the hand pose and grasp: $Grasp_{o,r}(H, G)$, where the test on the variables H and G is whether, if the robot hand is at H holding o with grasp G, then o is contained in the r.

In a CSP, constraints involving small numbers of variables can often be used early to achieve significant pruning. The $Grasp$ constraint involves several variables, so we add some conservative low-arity constraints to allow early pruning: given the desired pose or region for an object, we can determine simple unary "box" constraints on the x, y, z, and θ coordinates of the hand that rule out hand poses that are not feasible for any possible value of G. These constraints are necessary but not sufficient, and so we must include the high-arity high-fidelity original grasp constraints as well.

[1]In our experiments we test cases with 40 and 120 values for K.

```
FREESPACEMAP(Θ, bCS, hCS, gCS)
 1  bM = INITBASEMAP(Θ)
 2  for tr ∈ Θ
 3      for (k, cs) ∈ bCS
 4          UPDATE(CO(cs[tr], [0,0]), bM[tr], k)
 5  Z = PICKZRANGES(hCS + gCS)
 6  fsM = INITFSMAP(Θ, Z)
 7  for tr ∈ Θ
 8      for zr ∈ Z
 9          fsM[zr][tr] = bM[tr]
10          for (k, cs) ∈ hCS
11              UPDATE(CO(cs[tr], zr), fsM[tr][zr], k)
12          for (g, cs) ∈ gCs
13              UPDATE(CO(cs[tr], zr), fsM[tr][zr], g)
14  return fsM
```

Fig. 1: Computing the free space map.

The constraint that the robot have collision-free paths is much more difficult. We enforce an implicit constraint that all configurations of the robot, at pick and place locations, as well for the paths to and from the home configuration are all in the same connected component of the robot free space.

The ∃VP constraint additionally requires checking for paths through the free configuration space, possibly with some additional obstacles. We cannot afford to call a motion planner each time that we need to evaluate such a constraint during the solution process. Instead, we perform significant pre-computation that renders the individual constraint-checks done during the search relatively efficient. The pre-computation is done in two phases: finding the connected component of the free configuration space that contains the robot and then finding paths through that space that satisfy additional constraints. We explore two versions of this: one in which the free-space map takes into account all parts of the robot, and one in which it only takes into account the base and the hand (but not the arm). In the second case, some additional collision-checking must be done during the CSP process, but the pre-computation time is significantly reduced.

We pre-compute a free space "map" for the robot (the valid values of H, K and G) by explicitly computing polygonal C-space "slices" for the environment objects at fixed values of H_z and for fixed ranges of H_θ [1]. Concretely, a free-space map is an array, indexed by entries from $\mathcal{H}_\theta$ (a range of angles) and $\mathcal{H}_z$ (a range of z values) whose entries are x, y maps; each map is represented as a vector indexed by values in $\mathcal{H}_y$ of ranges of x values in the free-space of the hand. Each range of x is tagged with the set of K and G values for which it is free. Pseudocode is shown in figure 1.

The inputs, bCS, hCS and gCS, are lists of pairs: the first element of each pair is a value of value of K or G; the second element, denoted by cs, is a vector indexed by a range of θ (Θ is a set of such θ ranges) of lists of polyhedral (x, y, z) C-space obstacles for either the base, the arm (and hand) or a grasped object. Figure 2 illustrates how these polyhedra are obtained. C-space obstacles are computed for the three robot components: the base, the arm and hand, and the grasped

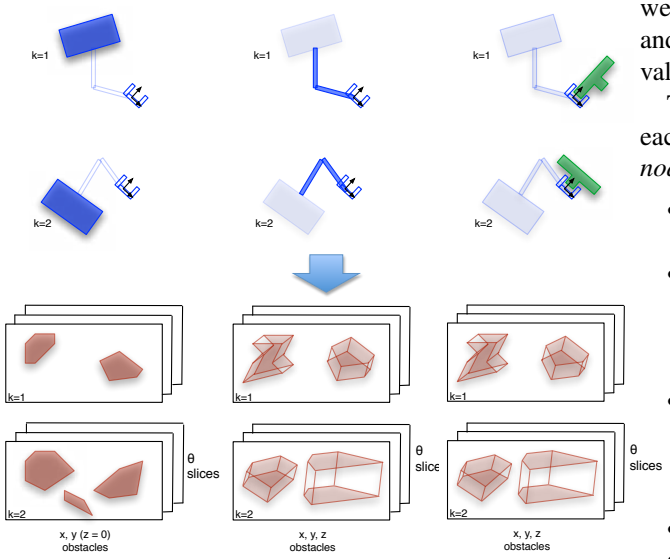

Fig. 2: Construction of free-space map. Consider two different kinematic arrangements, $k = 1$ and $k = 2$. For each one, compute configuration-space obstacles for the base; they are represented with continuous x values, sampled y values, z values dictated by the object shapes, and sampled θ values; shown schematically in bottom left. Repeat for arm (middle) and hand, potentially holding an object (right). The free space map is the complement of the union of these obstacles.

object—all defined relative to same reference frame, in this case, the hand frame—interacting with each of the obstacles. One subtlety is that the base map only has a single z range (for $z = 0$) since the base stays on the ground.

The procedure CO takes such a polyhedral map and a range of z values and produces a list of x, y C-space polygons. These polygons are then scan-converted and merged into an existing map by the UPDATE procedure. The base maps (for each range of θ) are used to initialize all the maps for the hand/arm, since a collision for the base blocks access for any hand value of z.

Having this map enables efficient checks for the safety of individual robot configurations. Some of the cost of building this representation can be amortized over multiple calls to the CSP solver in closely related environments.

Each ∃VP constraint specifies some static conditions on the search: which objects are at their initial poses, $\mathcal{O}_I$, and which object is in the hand, o_G, which may be $\emptyset$. There are no more of these search conditions than there are steps in the plan, and they can be straightforwardly extracted from the plan skeleton. For each of these search conditions, we construct a spanning tree, rooted at the home configuration, representing the connectivity of the configuration space cells.

Note that the values of K and G are typically independent; that is, collisions of the grasped object with other objects can be checked independently of collisions of the robot with other objects. In principle, there can be "self-collisions" between the grasped object and the robot; we choose values of K and G to ensure that this does not happen. Therefore, we represent tree nodes for the discretized hand parameters and at each node represent which values of K and G are valid at that node.

The tree of paths is represented as a collection of nodes, each of which points to its parent in the tree. A *path-tree node* for $\mathcal{O}_I, o_G$ contains the following information:

- H: A discrete cell in the discretized hand configuration space.
- $\mathcal{K}_v$: The set of valid K (mappings from hand to base poses), such that the robot at configuration H, K is not in collision with any permanent obstacles or with any $O \in \mathcal{O}_I$.
- $\mathcal{G}_v$: The set of valid G (mappings from hand to poses of the grasped object) such that if o_G is in relation G to H, then it is not in collision with any permanent obstacles or with any $O \in \mathcal{O}_I$.
- c: The path cost, which is simply measured in cells.
- ϕ: A pointer to the parent node in the path tree

These trees pre-compute paths that avoid objects that are placed at their initial conditions; but we will also need to be sure that objects that are placed *during* the plan do not conflict with the robot's paths. It would be computationally much too expensive to pre-compute paths that avoid all possible locations of the objects that are placed during the execution of the plan skeleton. In our implementation, we make a simplifying assumption: when evaluating a constraint, we check whether the specified object placements collide with the pre-computed path to the target location. This means that there are situations where we will fail to find a solution even though one may exist in the quantized C-space, although we believe that these are in rare in practice.

The procedure EVPEVAL, in figure 3 evaluates an ∃VP constraint with respect to V, which is an assignment of values to constraint variables. The parameters $Hvar$, $Kvar$, and $Gvar$ are the variables (not values) to which this constraint applies; o_G is the grasped object, $\mathcal{O}$ is the set of objects to be avoided (both those that are known to be at their initial conditions and those that will have been placed during a previous plan step), and T is the set of path trees.

We begin, in lines 1–3 by finding which objects are placed at their initial conditions and using those and the grasped object to get the tree τ that corresponds to those planning conditions. In line 4, we look to see if the bound values for H and K are represented in some node in the tree. If not, then the home configuration is not reachable. Lines 7–11 constitute a loop that traverses the path from the target node back up to the root node. If the specified grasp value $V[Gvar]$ is not in the legal set of grasps for the node, or if the placed objects, at the places specified for them in V cause a collision with this node (which means that it collides[2] for *all* values of $K \in n.\mathcal{K}_v$), then we fail and return FALSE. If the entire path to the root is valid, we return TRUE.

Constraints used in our CSP solver must have fixed *arity*; that is, they involve a fixed set of constraint variables.

[2]When the free space map is built without considering the arm, we check for arm collisions at this state, inside the constraint test.

EVPEVAL$(Hvar, Kvar, o_G, Gvar, \mathcal{O}, V, T)$

$\mathcal{O}_I = getInitialObjects(\mathcal{O})$
$\mathcal{O}_P = getPlacedObjects(\mathcal{O})$
$\tau = T(\mathcal{O}_I, o_G)$
$n =$ FINDNODE$(V[Hvar], V[Kvar], \tau)$
if $n == \emptyset$
    return False
while $n.\phi \neq \emptyset$
    if $V[Gvar] \notin n.\mathcal{G}$ **or**
      COLLIDES$(n, \mathcal{O}_P, o_G, V[Gvar], V)$
        return False
    $n = n.\phi$
return True

Fig. 3: Valid path constraint test.

EVPEVAL operates on a set of placed objects $\mathcal{O}_P$, each of which is specified by H and G variables; thus, it does not have fixed arity. In practice, we introduce a separate $\exists$VP constraint for each placed obstacle.

C. Solving the CSP

We use an "off-the-shelf" CSP solver developed by Chen and Van Beek [23]. It uses a combination of backtracking (with backjumping), and some combination of propagation, ranging from forward checking to full arc-consistency. One can choose the level of propagation per constraint; we choose limited propagation for the $\exists$VP constraints and full propagation for the others.

VII. RESULTS

We tested the performance of several variations of the proposed approach in five scenarios, using a pilot implementation in unoptimized Python. The relative performance of the different methods is well characterized, but the timings are incomparable to more optimized implementations of other approaches.

A. Scenarios

The first four test scenarios correspond to finding plans for the four skeletons discussed in section V. Figure 4 shows the solution to a relatively difficult regrasping problem. Figure 5 shows an object placement in a more complex domain. Results for the other two scenarios are available in the accompanying video material.

The last scenario (shown in the accompanying video material) corresponds to an entire execution of the task planner, with the goal of putting an object into a target region, which requires moving two other objects out of the way in the process. During the course of the task-level planning, it has to do the pre-processing once, but then calls the CSP 35 times; 34 of these calls involve plan skeletons whose constraints are not satisfiable. This is a realistic situation, highlighting how critical it is for an approach to be able to fail effectively as well as to succeed. The time to execute

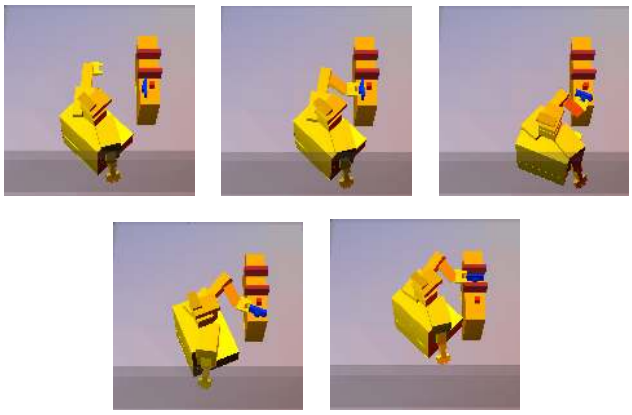

Fig. 4: Given a plan skeleton consisting of pick, place, pick, and place operations all on the same object and the constraint that the object end up in a certain region, the CSP determines grasps, object placements, robot configurations, and paths that satisfy the constraints.

Fig. 5: Plan to pick an object from an awkward location on one table, and move it to the back table.

the entire search, including all 35 calls to the CSP was 192 seconds.

B. Comparing algorithms

In this section we present a quantitative comparison of several different variations on the general approach. We consider the following algorithms:

- BT-Sim : pure backtracking (no constraint propagation) using simplified configuration-space in pre-processing (not including the arm)
- CSP-Sim: constraint satisfaction using simplified configuration-space
- CSP-Full: constraint satisfaction using complete all configuration space obstacles
- CSP-RRT: constraint satisfaction, with no pre-processing, but calling an RRT planner in the test for the $\exists$VP constraints

For each of these cases, we tried two values of several complexity parameters:

- Number of obstacles in the environment: $n_o = 5, 15$
- Number of grid samples of y coordinate: $n_y = 64, 128$
- Number of grid samples of θ: $n_\theta = 5, 12$
- Number of kinematic solutions: $n_k = 40, 120$

Recall that we do not discretize the x coordinate. In our experiments, we use object models that are unions of extrusions in z; that is, objects with a constant z cross-section. So, z values are sampled at every value where the collection of

n_k	n_θ	n_y	n_o	Pre	BT F	CSP F	BT I	CSP I
40	5	64	5	10.2	23.3	5.5	59.3	0.0
40	5	64	15	15.2	–	–	9.4	0.0
40	5	128	5	17.9	197.2	13.2	123.5	0.0
40	5	128	15	26.8	–	–	19.3	0.0
40	12	64	5	24.6	25.9	9.2	61.6	2.9
40	12	64	15	36.0	–	–	9.6	0.4
40	12	128	5	47.0	275.3	30.6	128.7	4.8
40	12	128	15	65.9	–	–	21.8	0.8
120	5	64	5	48.0	74.4	17.3	335.7	0.0
120	5	64	15	63.8	2.0	1.7	67.7	0.0
120	5	128	5	99.4	258.4	24.5	500.0	0.0
120	5	128	15	123.8	335.3	9.7	132.0	0.0
120	12	64	5	105.2	78.7	22.3	356.6	2.9
120	12	64	15	149.5	3.6	2.9	78.9	4.7
120	12	128	5	216.9	327.5	79.5	500.0	12.4
120	12	128	15	302.8	336.7	13.8	154.3	1.8

TABLE III: Run-times in seconds on problems described in the text. Columns correspond to: BT F is backtracking on feasible problems; CSP F is CSP on feasible problems; BT I is backtracking on infeasible problems; CSP I is CSP on infeasible problems. An entry of '–' means that the problem was feasible but no solution was found, due to coarse granularity of discretization. A value of 0.0 means it ran less than 0.05 seconds. Jobs were terminated after 500 seconds; times for such jobs reported as 500s.

objects changes cross section. In these tests we use relatively simple objects that have on the order of four possible grasps.

We ran each algorithm under each of the 16 complexity conditions on each of the four plan skeletons, and report averages over the skeletons. Table III shows the time taken for pre-processing and the time taken to find a solution after pre-processing for BT-Sim and CSP-Sim. Backtracking runs were terminated after 500 seconds.

The CSP-Full and CSP-RRT algorithms are not competitive with the others. CSP-Full frequently runs out of memory due to the complexity of the configuration space it has to represent. CSP-RRT takes about two orders of magnitude longer to run (using a relatively naive Python implementation of the RRT). Because the algorithm never explicitly fails, it requires that a time-out be selected, but it is well known that running times on feasible problems have high variance, so it is difficult to select a time-out value. We did some preliminary experiments using a probabilistic road-map approach, in which the road map is constructed at pre-processing time, and shared by all calls to the $\exists$VP constraint test. This approach is very promising and might equal the performance of CSP-Sim with less pre-processing.

These results show the consistent advantage of the CSP method over pure backtracking. However, the behavior on feasible problems does not begin to tell the full story. In the context of task planning, the time taken to fail is even more important than the time taken to succeed. We attempted to find a plan using a skeleton with just one pick and one place operation in a domain that required moving something out of the way. In Table III we report the time taken to fail, by BT-Sim and CSP-Sim. Note that the CSP is dramatically faster than BT in detecting failures. It is interesting to note that BT fails faster in the more cluttered environments, presumably because it fails earlier in the search tree.

The constraint-based formulation described here provides an attractive approach to integrating symbolic and geometric constraints for TAMP. One important drawback is the need to pick an arbitrary discretization; we are investigating alternative approaches that generate task specific discretizations.

REFERENCES

[1] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, pp. 681–698, 1981.

[2] T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O'Donnell, W. E. L. Grimson, P. Tournassoud, and A. Lanusse, "Handey: A robot system that recognizes, plans, and manipulates," in *ICRA*, 1987, pp. 843=–849.

[3] G. T. Wilfong, "Motion planning in the presence of movable obstacles," in *Symposium on Computational Geometry*, 1988, pp. 279–288.

[4] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *IJRR*, vol. 23, pp. 729–746, 2004.

[5] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *IJRR*, vol. 30, no. 6, pp. 676–698, 2011.

[6] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," in *Proceedings of Robotics: Science and Systems*, 2011.

[7] J. Barry, K. Hsiao, L. Kaelbling, and T. Lozano-Pérez, "Manipulation with multiple action types," in *Int. Symp. on Experi. Robotics*, 2012.

[8] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *IJRR*, vol. 28, 2009.

[9] E. Plaku and G. Hager, "Sampling-based motion planning with symbolic, geometric, and differential constraints," in *ICRA*, 2010.

[10] K. Hauser, "Randomized belief-space replanning in partially-observable continuous spaces," in *Workshop on Algorithmic Foundations of Robotics*, 2010.

[11] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE Conference on Robotics and Automation*, 2011.

[12] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *ICRA*, 2011.

[13] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *IROS*, 2012.

[14] S. Srivastava, L. Riano, S. Russell, and P. Abbeel, "Using classical planners for tasks with continuous operators in robotics," in *ICAPS Workshop on Planning and Robotics (PlanRob)*, 2013.

[15] C. Erdogan and M. Stilman, "Planning in constraint space: Automated design of functional structures," in *ICRA*, 2013.

[16] R. Dechter, *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[17] T. Lozano-Pérez, J. L. Jones, E. Mazer, and P. A. O'Donnell, *Handey: A Robot Task Planner*. Cambridge, MA: MIT Press, 1992.

[18] R. Alami, T. Simeon, and J.-P. Laumond, "A geometrical approach to planning manipulation tasks. the case of discrete placements and grasps," in *ISRR*, 1990.

[19] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe, "Planning motions with intentions," in *SIGGRAPH*, 1994.

[20] R. Haralick and G. Elliot, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, no. 3, pp. 263–313, 1980.

[21] A. K. Pandey, J.-P. Saut, D. Sidobre, and R. Alami, "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*, 2012.

[22] M. Stilman and J. J. Kuffner, "Planning among movable obstacles with artificial constraints," in *Proceedings of the Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2006.

[23] X. Chen and P. van Beek, "Conflict-directed backjumping revisited," *J. Artif. Intell. Res. (JAIR)*, vol. 14, pp. 53–81, 2001.

[24] K. Hauser, "The minimum constraint removal problem with three robotics applications," in *WAFR*, 2012.