States Path Finder: a general and efficient algorithm for prehensile manipulation planning

Quang Anh Le, Alexandre Thiault, Florent Lamiraux

States Path Finder: a general and efficient algorithm for prehensile manipulation planning

Quang Anh Le
SCSE
Nanyang Technological University
Singapore
quanganh001@e.ntu.edu.sg

Alexandre Thiault
École Polytechnique
Institut Polytechnique de Paris
Paris, France
alexandre.thiault@polytechnique.edu

Florent Lamiraux
LAAS-CNRS
University of Toulouse
Toulouse, France
florent.lamiraux@laas.fr

***Abstract*—This paper proposes a new algorithm called States Path Finder (SPF) to solve the general prehensile manipulation problem for multiple robots and multiple objects. Using a constraint graph to represent the problem, SPF finds a list of transitions from the initial state to the goal state, computes a corresponding set of waypoints and connects the waypoints to form a continuous, collision-free solution path. We benchmark SPF on several problem instances with varying characteristics. The results suggest SPF's generality and also its higher efficiency in comparison to Manipulation-RRT, an existing RRT-like algorithm also based on the constraint graph.**

***Index Terms*—Manipulation planning, motion planning, prehensile manipulation**

I. Introduction

Manipulation planning is a class of problems where some robots act on some objects in order to displace them from an initial to a goal pose. The action is usually performed through grasping or pushing. In both cases, motion is achieved by contact. Contacts impose constraints on the configuration of the whole system (robot + obstacles). These constraints can be expressed numerically. They define submanifolds of the configuration space. The set of admissible configurations is thus the union of such submanifolds, depending on the constraints that are active. For instance, when an object is not grasped, it should lie in a stable pose, when it is grasped it should follow the gripper that grasps it.

Manipulation planning, also called task and motion planning (TAMP), is often broken down into more specific instances.

Rearrangement planning [1]–[4] for example is a class of task and motion planning problems where one or several robots are requested to move some objects from an initial to a final pose. Some papers handle the specific case of multi-arm path planning [5]–[7], while other papers address the problem called navigation among movable obstacle (NAMO) where the goal is for the robot to reach a goal configuration, no matter the final pose of movable objects [8]–[10].

Sampling based path planning methods have been first applied to the manipulation planning problem in [11]. This pioneering work mainly addresses pick and place problems. In the first step, the objects are allowed to slide on the contact surface while grasped, which makes the search more efficient. In the second step, a reduction property enables the algorithm to approximate the path found in the first step by a valid manipulation path. Later, [12] propose an extension of Rapidly-Exploring Random Trees (RRT) to the problem of a humanoid robot walking to a table and pushing an object to a goal region. The RRT algorithm explores the configuration space as well as a graph of modes representing the different states of the system (walking, moving the arm, pushing the object). They prove the probabilistic completeness of their approach. [13] builds on this previous work to propose a method that is asymptotically optimal. They validate the algorithm on a simple example where planar robots push planar objects to a goal zone. In a previous paper [14] however, we showed that naively running an RRT algorithm exploring the modes of the manipulation problem fails in some cases. Namely, if two foliated manifolds intersect, RRT will never be able to connect trees in different manifolds. This issue arises in cases as simple as a robot manipulating two objects that can be put at any place on a horizontal plane. In this latter case, the placement manifold of each object is foliated: each pose of the object on the plane defines a leaf (mode in [12], orbit in [13]).

Among the few works that properly handle crossed foliations, [15] proposes a method for the simple case of a single manipulator arm manipulating a single object. The method builds roadmaps on some leaves of the manifolds *grasp* and *placement* by running PRM*, then it connects the roadmap by sampling configurations at the intersection of the leaves containing roadmaps.

In this paper, we propose an extension of the latter method to more general prehensile manipulation problems with several objects and several robots.

II. Manipulation planning problem

In this section, we define the class of manipulation planning problems we address. We recall the main concepts and notation from [16].

This work has been partially supported by the Joint lab ROB4FAM between Airbus and the CNRS

A. Definitions

Definition 1: **Prehensile manipulation problem**
A prehensile manipulation problem is defined by

- $nr \geq 1$ robots, $no \geq 1$ objects,
- a set of possible grasps,
- environment contact surfaces,
- object contact surfaces,
- an initial configuration $\mathbf{q}_{init}$,
- a final configuration $\mathbf{q}_{goal}$.

The *Configuration space* of the system is the Cartesian product of the robot and object configuration spaces:

$$\mathcal{C} = \mathcal{C}_{r_1} \times \cdots \times \mathcal{C}_{r_{nr}} \times SE(3)^{no}$$

Admissible configurations of the system are configurations that satisfy the following property:

- each object is either grasped by a robot, or lies in a stable contact pose,
- the volumes occupied by the links of the robots and by the objects are pair-wise disjoint.

Admissible motions of the system are motions that satisfy the following property:

- configurations along the motion are admissible, and
- the pose of objects not grasped is constant,
- the relative pose of objects grasped by a gripper with respect to the gripper is constant.

The solution of a prehensile manipulation problem is an admissible motion that links the initial and goal configurations.

Definition 2: A **numerical constraint** is defined by a piecewise C^1 mapping h from $\mathcal{C}$ to a vector space $\mathbb{R}^p$ for some positive integer p, and by a right hand side $\mathbf{h}_0 \in \mathbb{R}^p$. A configuration $\mathbf{q} \in \mathcal{C}$ is said to satisfy the constraints iff

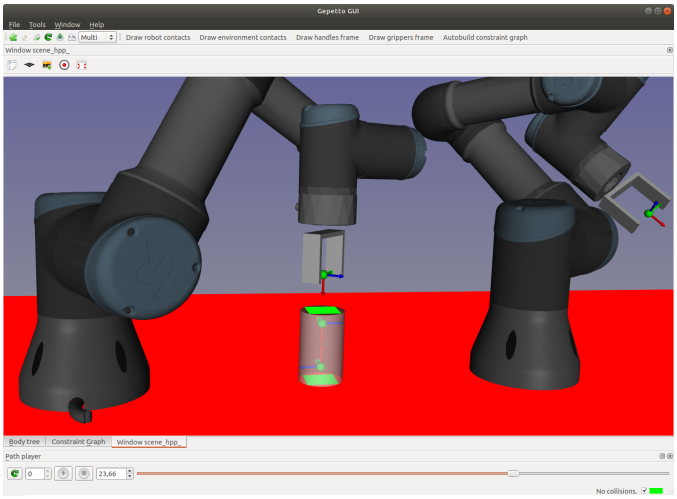
$$h(\mathbf{q}) = \mathbf{h}_0$$

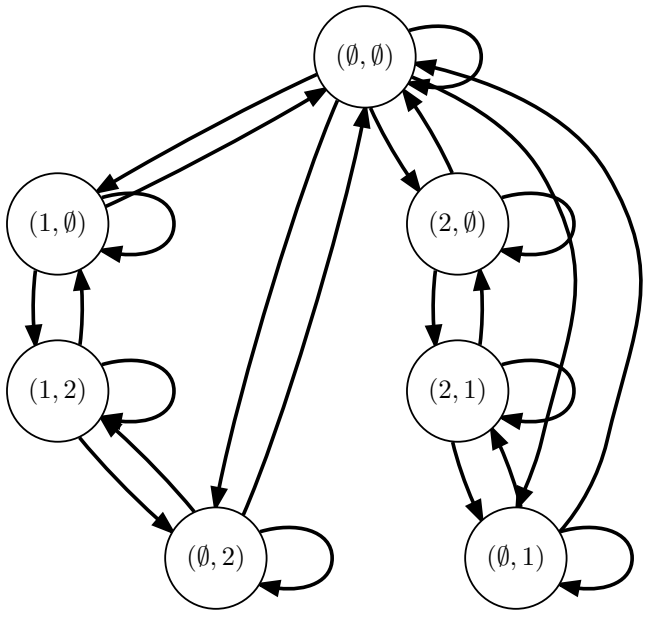
Definition 3: **Grasp** Given a frame attached to a robot link called *gripper* and a frame attached to an object called *handle*, a grasp is the numerical constraint defined by the $\mathbb{R}^6$-valued mapping g such that if $\mathfrak{g}(\mathbf{q}) \in SE(3)$ and $\mathfrak{h}(\mathbf{q}) \in SE(3)$ respectively denote the pose of the gripper and handle in configuration $\mathbf{q}$,

- the first 3 components of $g(\mathbf{q})$ represent the coordinates of the origin of $\mathfrak{h}(\mathbf{q})$ in the frame $\mathfrak{g}(\mathbf{q})$,
- the last 3 components of $g(\mathbf{q})$ represent a vector (expressed in $\mathfrak{g}(\mathbf{q})$) whose axis is the axis of the rotation that drives frame $\mathfrak{g}(\mathbf{q})$ to $\mathfrak{h}(\mathbf{q})$ and whose norm is the angle of the latter rotation.

Note that $g(\mathbf{q}) = 0$ implies that frames $\mathfrak{g}$ and $\mathfrak{h}$ coincide.

Definition 4: **Placement** Given some convex polygons attached to the environment and some convex polygons attached to an object, the object is said to be in placement if the center of one of the object polygons is included in the surface delimited by one of the environment polygons and the outward normals of the polygons are opposite to each other.
It is possible to express placement as a numerical constraint as explained in [16], Section V.B.





Constraint graph

Fig. 1: Example of manipulation planning problem. Top: two UR3 robots with one gripper each (X=red, Y=green, Z=blue) manipulating a cylinder with two handles. The environment contains one rectangular contact surface (in red). The cylinder has two rectangular contact surfaces (in green). Bottom: the corresponding constraint graph. Names of states follow Expression (5): for example, $(\emptyset, 1)$ means that gripper of robot 2 grasps handle 1 of the cylinder. In this state, there is no placement constraint.

B. Constraint and complement

A placement constraint pl only partially constrains the pose of an object. It is possible to define a placement complement constraint $\bar{pl}$ in such a way that along a transit path γ where the object is not grasped:

$$pl(\gamma(t)) = 0 \quad (1)$$
$$\bar{pl}(\gamma(t)) = \bar{pl}(\gamma(0)) \quad (2)$$

The second constraint ensures that the object remains static along transit motions. See [16], Section V.B. for details.

Similarly, a grasp constraint gr can be defined by only selecting some components of function g in Definition 3. For instance selecting only the first 5 components, the grasp constraint allows a rotation around the z-axis of the gripper. In this case, the components that are dropped need to be constant along transfer motions were the gripper moves the object. If gr denotes the mapping with only some components selected, we define $\bar{gr}$ as the mapping defined on $\mathcal{C}$ gathering the other components. Then, along a transfer motion $\gamma : [0,1] \to \mathcal{C}$ the following constraints hold for all t in $[0,1]$:

$$gr(\gamma(t)) = 0 \quad (3)$$

$$\bar{gr}(\gamma(t)) = \bar{gr}(\gamma(0)) \quad (4)$$

C. Constraint graph

The set of admissible configurations is a union of submanifolds defined by grasp and placement constraints. We represent this set as a graph called *constraint graph*. Let $n_{\mathfrak{g}}$ be the number of grippers and $n_{\mathfrak{h}}$ be the number of handles. We represent a state as a sequence of $n_{\mathfrak{g}}$ indices

$$(h_1, \cdots, h_{n_{\mathfrak{g}}}), \quad (5)$$

where $h_i \in \{\emptyset, 1, \cdots, n_{\mathfrak{h}}\}$, $1 \leq i \leq n_{\mathfrak{g}}$. $h_i = j$ for $1 \leq j \leq n_{\mathfrak{h}}$ means that gripper i grasps handle j. $h_i = \emptyset$ means that gripper i does not grasp anything. The numerical constraints associated to a state are therefore

- constraints (3) for each active grasp,
- constraints (1) for each object that is not grasped.

Two states are said to be adjacent if they are equal or if they differ by only one grasp and the grasp is empty in one of thoses states. The nodes of the constraint graph are the states as defined above and the edges link adjacent states. Edges store the constraints of paths that link configurations in their starting and destination states. Let S_1 and S_2 be two adjacent states with S_1 being the state with one grasp less than S_2. The constraints of the edges linking S_1 to S_2 and S_2 to S_1 are

- the constraints of state S_1,
- the grasp complement constraints (4) of each grasp in S_1,
- the placement complement constraints (2) for each object not grasped in S_1.

Edges of the constraint graph represent foliated manifolds. Two configurations belong to the same leaf if and only if the right hand sides of their complement (placement and grasp) constraints are all equal. Figure 1 illustrates the various concepts introduced in this section. In the following sections, nodes and edges of the constraint graph are called states and *transitions* to avoid confusion with nodes and edges of roadmaps. Transitions are denoted by $\mathcal{T}$. The states they link are denoted by $\mathcal{T}$.from and $\mathcal{T}$.to.

Note that some states may be a subset of others. For instance, in Figure 1, state $(2,1)$ is included in $(2,\emptyset)$ and $(\emptyset,1)$. We denote as $\mathcal{S}$ the mapping from $\mathcal{C}$ to the set of states that maps to each configuration $\mathbf{q}$ the smallest state that $\mathbf{q}$ belongs to, or $\emptyset$ if $\mathbf{q}$ does not belong to any state.

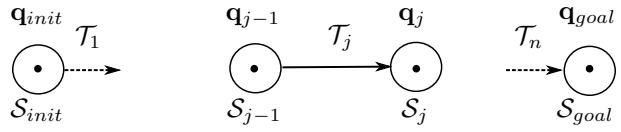


Fig. 2: Waypoints are randomly sampled in consecutive states.

III. States Path Finder

In this section, we describe our manipulation planning algorithm. The algorithm iterates over the following actions:

1) find a sequence of transitions between the states of the initial and goal configurations,
2) sample configurations called *waypoints* in the successive destination states of those transitions,
3) link these waypoints using a bi-RRT* algorithm in the leaves containing 2 successive waypoints.

A. Sequence of transitions

Let $\mathcal{S}_{init} = \mathcal{S}(\mathbf{q}_{init})$ and $\mathcal{S}_{goal} = \mathcal{S}(\mathbf{q}_{goal})$ be the states containing $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ respectively. We extract all the paths in the constraint graph from $\mathcal{S}_{init}$ to $\mathcal{S}_{goal}$ in order of increasing length. We denote by

$$seq = (\mathcal{T}_1, \cdots, \mathcal{T}_n)$$

any sequence such that $\mathcal{T}_1$.from $= \mathcal{S}_{init}$, $\mathcal{T}_n$.to $= \mathcal{S}_{goal}$, $\mathcal{T}_j$.to $= \mathcal{T}_{j+1}$.from for any integer j between 1 and $n-1$.

Figure 2 gives an example of sequence of transtions.

B. Sampling of waypoints

Given a sequence of transitions as described in the previous section, we will randomly sample $n-1$ configurations in states $\mathcal{T}_j$.to for j between 1 and $n-1$. To sample a configuration on a manifold defined by some constraints, we shoot a random configuration that we give as the initial guess to a Gauss-Newton solver [17, chapter 10]. The solver will perform a *projection* onto the manifold.

Let $\mathbf{q}_0 = \mathbf{q}_{init}$ and $\mathbf{q}_n = \mathbf{q}_{goal}$, and let $\mathbf{q}_j$ for j in $\{1, \cdots, n-1\}$ denote the waypoints. We also denote by $\mathcal{S}_{j-1}$ and $\mathcal{S}_j$ the initial and destination states of transition $\mathcal{T}_j$.

Let $h_{\mathcal{S}_j}$ and $h_{\mathcal{T}_j}$[1] denote the constraints associated to state $\mathcal{S}_j$ and transition $\mathcal{T}_j$ respectively. Then, each waypoint should satisfy the constraints of the state it is contained in and of the transition that leads to this state. Namely:

$$h_{\mathcal{S}_j}(\mathbf{q}_j) = 0 \quad (6)$$

$$h_{\mathcal{T}_j}(\mathbf{q}_j) = h_{\mathcal{T}_j}(\mathbf{q}_{j-1}) \quad (7)$$

$$h_{\mathcal{T}_n}(\mathbf{q}_{n-1}) = h_{\mathcal{T}_n}(\mathbf{q}_{goal}) \quad (8)$$

Equation (6) states that waypoint $\mathbf{q}_j$ is in state $\mathcal{S}_j$. Equation (7) implies that between $\mathbf{q}_{j-1}$ and $\mathbf{q}_j$, objects that are not grasped

[1]Note that $h_{\mathcal{S}_j}$ and $h_{\mathcal{T}_j}$ are actually composed of several constraints of types (1-4).

are in the same stable pose, and object that are grasped keep the same relative pose with respect to the gripper that grasps them.

1) Computing right hand sides: We could naively solve these nonlinear systems of equations iteratively from $j = 1$ to $j = n - 1$ and use the result of an iteration to initialize the right hand side of (7) at the next iteration. However, doing so would most likely generate a $\mathbf{q}_{n-1}$ that does not satisfy (8). Some right hand sides need to be initialized using $\mathbf{q}_{goal}$. We therefore propose the following method. We define a matrix RHS with $n - 1$ columns corresponding to each waypoint, and n_c lines where n_c is the number of different constraints of types (1-4) in all transitions. We refer to h_i as the piecewise C^1 mapping defining constraint number i for i between 1 and n_c. Algorithm 1 computes for each constraint and each waypoint, how the right hand side should be initialized. Note that line 14 of this algorithm detects sequences of transitions that cannot give rise to a valid sequence of waypoints.

2) Computing waypoints: Once matrix RHS has been computed, we compute the waypoints iteratively from $\mathbf{q}_1$ to $\mathbf{q}_{n-1}$. For each $\mathbf{q}_j$, the function INITIALIZESOLVER (line 9) creates a solver and inserts the constraints that have been initialized in column j of matrix RHS. We initialize the right hand side of each constraint i at step j as follows:

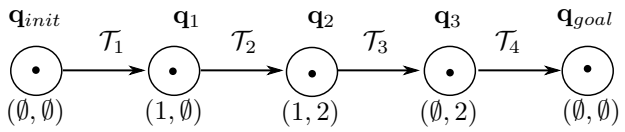
$$h_i(\mathbf{q}) \quad = \quad \begin{cases} h_i(\mathbf{q}_{init}) & \text{if} \quad RHS_{i,j} = \text{EqualToInit} \\ h_i(\mathbf{q}_{goal}) & \text{if} \quad RHS_{i,j} = \text{EqualToGoal} \\ h_i(\mathbf{q}_{j-1}) & \text{if} \quad RHS_{i,j} = \text{EqualToPrevious} \end{cases}$$

Algorithm 1 Right hand side initialization

```
1:  for i from 1 to n_c do
2:      for j from 1 to n − 1 do
3:          if h_i in T_j then
4:              if j = 1 or RHS_{i,j−1} =EqualToInit then
5:                  RHS_{i,j} ← EqualToInit
6:              else
7:                  RHS_{i,j} ← EqualToPrevious
8:      for j from n − 1 down to 1 do
9:          if h_i in T_j then
10:             if j = n−1 or RHS_{i,j+1} =EqualToGoal then
11:                 if RHS_{i,j} =EqualToInit then
12:                     if h_i(q_init) = h_i(q_goal) then
13:                         break
14:                     else return false
15:                 else RHS_{i,j} =EqualToGoal
```

Figures 3 shows an example of computation of a sequence of waypoints.

Algorithm 2 explains how waypoints are computed. For each waypoint, Function SOLVESTEP shoots a random configuration and solves the corresponding system of equations using Gauss-Newton solvers (line 2), and checks for collision (line 3). If the resolution is successful, and the resulting configuration $\mathbf{q}$ is collision free, we store the valid configuration and we switch to the next waypoint (line 17). If the number of calls to Function SOLVESTEP for a given waypoint exceeds a local threshold N_{local} (line 12), the algorithm backtracks to the previous waypoint (line 14). To ensure the algorithm halts when given an infeasible list of transitions, the total number of failures is stored and checked against the global threshold N_{global} (line 18). When the algorithm fails to solve all waypoints, it will return the longest sequence of valid waypoints so far.



Constraints	from
$\mathbf{q}_1$	
$pl(\mathbf{q}) = 0$	$(\emptyset, \emptyset) \rightarrow (1, \emptyset)$
$\bar{pl}(\mathbf{q}) = \bar{pl}(\mathbf{q}_{init})$	$(\emptyset, \emptyset) \rightarrow (1, \emptyset)$
$gr_{(1,\emptyset)}(\mathbf{q}) = 0$	$(1, \emptyset) \rightarrow (1, 2)$
$\mathbf{q}_2$	
$gr_{(1,\emptyset)}(\mathbf{q}) = 0$	$(1, \emptyset) \rightarrow (1, 2)$
$\bar{gr}_{(1,\emptyset)}(\mathbf{q}) = \bar{gr}_{(1,\emptyset)}(\mathbf{q}_1)$	$(1, \emptyset) \rightarrow (1, 2)$
$gr_{(\emptyset,2)}(\mathbf{q}) = 0$	$(1, 2) \rightarrow (\emptyset, 2)$
$\mathbf{q}_3$	
$gr_{(\emptyset,2)}(\mathbf{q}) = 0$	$(1, 2) \rightarrow (\emptyset, 2)$
$\bar{gr}_{(\emptyset,2)}(\mathbf{q}) = \bar{gr}_{(\emptyset,2)}(\mathbf{q}_2)$	$(1, 2) \rightarrow (\emptyset, 2)$
$pl(\mathbf{q}) = 0$	$(\emptyset, 2) \rightarrow (\emptyset, \emptyset)$
$\bar{pl}(\mathbf{q}) = \bar{pl}(\mathbf{q}_{goal})$	$(\emptyset, 2) \rightarrow (\emptyset, \emptyset)$

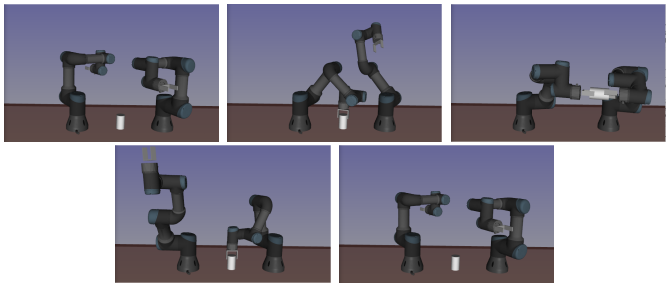


Fig. 3: Example of a sequence of 4 transitions for the system in Figure 1. The table shows the constraints inserted in each solver. The left column shows how the right hand sides are initialized. The right column shows which transition each constraint comes from. $gr_{(1,\emptyset)}$ and $gr_{(\emptyset,2)}$ are the numerical constraints defining grasps of handle 1 by gripper 1 and of handle 2 by gripper 2 respectively. These constraints follow Definition 3 where the 4th component of g is removed. This means the rotation around the x-axis of the gripper is free when grasping the object. As explained in Section II-B, $\bar{gr}_{(1,\emptyset)}$ and $\bar{gr}_{(\emptyset,2)}$ are the 1-dimensional constraints composed of the 4th component of the g function of Definition 3, and represent the angle of the rotation between the gripper and handle frames around the gripper x-axis. The five snapshots show $\mathbf{q}_{init}$, the 3 waypoints computed by solving the constraints and $\mathbf{q}_{goal}$. Between $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, the object is flipped upside down.

Algorithm 2 Solve waypoints

Compute collision-free waypoints $\mathbf{q}_i$ for i in $\{1, \cdots, n-1\}$. RHS is the right hand side matrix. N_{global} and N_{local} are global and local maximum failure thresholds.

function SOLVESTEP($solver$)
  $\mathbf{q} \leftarrow solver$.SOLVE()
  if solving succeeds and $isCollisionFree(\mathbf{q})$ **then**
    return $\mathbf{q}$
  return $\emptyset$
function SOLVEWAYPOINTS(n, RHS)
  Initialize $j \leftarrow 1$, $WP \leftarrow$ array of $n-1$ configs
  repeat
    $solver \leftarrow$ INITIALIZESOLVER(j, WP, RHS)
    repeat
      $\mathbf{q}, status \leftarrow$ SOLVESTEP($solver$)
    until $\mathbf{q} \neq \emptyset$ or failures exceeding N_{local}
    if $\mathbf{q} \neq \emptyset$ **then**
      $j \leftarrow j - 1$ ▷ Backtrack to previous waypoint
    else
      $WP[j] \leftarrow \mathbf{q}$
      $j \leftarrow j + 1$ ▷ Solve the next waypoint
  until $j = n$ or failures exceeding N_{global}
  return longest sequence of valid waypoints

3) Linking waypoints: Once a sequence of waypoints has been computed, we run a bi-RRT* between successive waypoints. Between waypoints $\mathbf{q}_{j-1}$ and $\mathbf{q}_j$ we apply constraints of transition $\mathcal{T}_j$. This means that all nodes and edges of the roadmap are projected onto the corresponding sub-manifolds.

IV. EXPERIMENTAL RESULTS

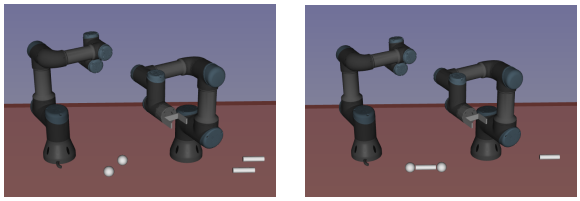
We benchmark States Path Finder (SPF) on multiple problem instances with different characteristics to test its generality and efficiency. The results are compared against Manipulation-RRT (M-RRT), our RRT-like algorithm proposed in a previous paper [14]. Both algorithms are based on the same representation of the problem as a constraint graph. The open source software Humanoid Path Planner [18] is used to implement the algorithms. The experiments are run on a desktop machine with 3.50GHz CPU and 8GB RAM. The full cpu and memory information, and the commit hashes of packages, are recorded for future reproducibility. Each benchmark is run 50 times for each algorithm. All parameters related to the performance of M-RRT and SPF are kept the same across all benchmarks.

M-RRT and SPF are compared on two criteria:

1) solving time: time taken to compute a solution path.
2) number of nodes: count all nodes in roadmaps generated by M-RRT and by the bi-RRT* inner planner of SPF. Generating a node in M-RRT is slightly more computationally expensive than in SPF's inner planner. Each time SPF calls bi-RRT*, it gives a single leaf to plan path on, and multiple nodes are generated on that leaf. Meanwhile, M-RRT is a full RRT planner and has additional computations to choose the leaf to project on.

A. Construction set

Two UR3 robot arms are supposed to assemble two spheres and a cylinder into a product (Figure 4). Each cylinder has two magnetic grippers that can grasp the magnetic handle of each sphere. The goal configuration specifies the relative pose between the cylinder and the spheres such that the only way to attach them is to grasp the sphere with one arm and the cylinder with another arm. If the cylinder is used to pick up any sphere from the table, it will have to grasp the sphere from below the table, which causes a collision.



(a) Initial config (b) Goal config

Fig. 4: *construction set*. Two UR3 robot arms assemble two spheres and a cylinder into a dumbbell-shaped product.

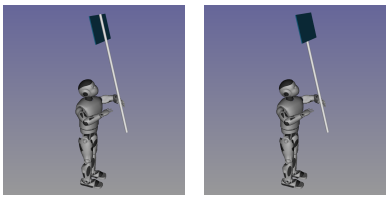
Table I shows the results of 50 runs of each algorithm. SPF performs much better in both criteria. The median solving time for SPF is more than 6 times smaller than M-RRT, while the median number of nodes is almost 4 times smaller.

Algo	Solving time (s)					Number of nodes				
	Min	Q1	Med	Q3	Max	Min	Q1	Med	Q3	Max
M-RRT	1.79	16.36	36.93	56.20	426.70	54.0	336.5	740.0	1040.8	9154.0
SPF	1.32	2.59	5.92	14.90	28.91	22.0	71.2	189.5	374.2	823.0

TABLE I: Minimum, maximum and quartiles for solving time and number of nodes over 50 runs on *construction set*.

B. Romeo placard

Romeo robot is supposed to rotate a placard by 180 degrees (Figure 5). The left hand can grasp a lower handle in the placard, while the right hand can grasp a higher handle. Both hands can hold onto the placard at the same time. The difficulty about this task is Romeo's highly detailed robot model, where each hand has 4 fingers with 3 segments [14]. A lot of time is required to project configurations and check collisions.



(a) Initial config (b) Goal config

Fig. 5: *Romeo placard*. Romeo must rotate a placard by 180°.

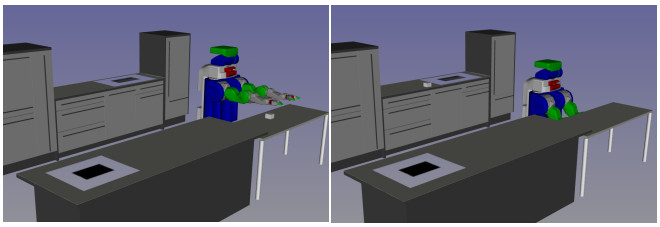
This benchmark continues to show superior performance for SPF (Table II). The median solving time for SPF is more than 30 times smaller than M-RRT, while the median number of nodes is more than 14 times smaller.

Algo	Solving time (s)					Number of nodes				
	Min	Q1	Med	Q3	Max	Min	Q1	Med	Q3	Max
M-RRT	2.77	61.97	115.03	190.94	378.31	21.0	291.5	510.0	960.5	1840.0
SPF	0.64	2.14	3.32	5.99	19.88	12.0	26.0	36.5	56.5	148.0

TABLE II: *Romeo placard* benchmark results.

C. PR2 manipulation kitchen

This is the only benchmark where $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ lie on different environment contact surfaces (Figure 6). Grasp is not foliated as there is only 1 way to grasp the box.



(a) Initial config (b) Goal config

Fig. 6: *PR2 manipulation kitchen*. PR2 robot must move a box from the dining table surface to the surface with the sink.

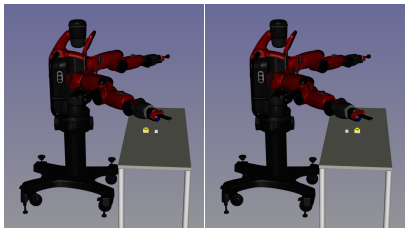
Results in Table III show no significant difference in performance between M-RRT and SPF. SPF median solving time is 1.6 times smaller than M-RRT, but its median number of nodes is slightly larger. The similar performance is likely due to non-foliated grasp. M-RRT does not face crossed foliation problem, removing one advantage of SPF over M-RRT.

Algo	Solving time (s)					Number of nodes				
	Min	Q1	Med	Q3	Max	Min	Q1	Med	Q3	Max
M-RRT	8.57	19.36	26.24	40.69	97.38	20.0	113.0	161.5	245.5	458.0
SPF	3.15	9.79	16.03	30.24	208.73	20.0	117.0	174.0	357.0	1239.0

TABLE III: *PR2 manipulation kitchen* benchmark results.

D. Baxter swaps two boxes

Baxter must swap two boxes on the table (Figure 7). Since Baxter has 2 arms, it can achieve this without having to place one box at an intermediate position on the table. Similar to *PR2 manipulation kitchen*, the boxes have non-foliated grasp.



(a) Initial config (b) Goal config

Fig. 7: *Baxter swaps two boxes*. Each box has one handle that can be grasped by either gripper.

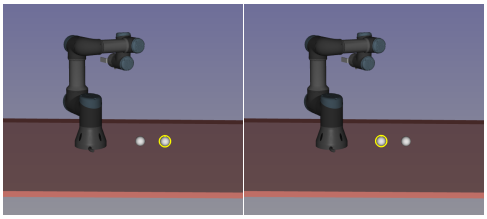
Results in Table IV display a better performance for SPF over M-RRT, but by a small margin. The median solving time for SPF is more than 2.3 times smaller than M-RRT. However, its median number of nodes is only 1.17 times smaller than M-RRT, which again suggests the effect of non-foliated grasp.

Algo	Solving time (s)					Number of nodes				
	Min	Q1	Med	Q3	Max	Min	Q1	Med	Q3	Max
M-RRT	0.39	0.89	1.70	2.20	5.91	17.0	35.2	65.5	82.5	234.0
SPF	0.35	0.47	0.73	0.92	2.61	15.0	24.2	56.0	85.5	452.0

TABLE IV: *Baxter swaps two boxes* benchmark results.

E. UR3 swaps two spheres

UR3 robot must swap the poses of two spheres on the table (Figure 8). Unlike Baxter robot with two arms, the main difficulty for UR3 is the need to generate at least one intermediate pose on the table to put one of the spheres.



(a) Initial config (b) Goal config

Fig. 8: *UR3 swaps two spheres* benchmark.

Table V shows the results. SPF median solving time is more than 11 times smaller than M-RRT, and its median number of nodes is more than 6 times. This benchmark is a case in which SPF is advantageous: the grasp is foliated, and the robot needs to generate a configuration with an intermediate placement pose that is not present in both $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, and still has to connect this configuration to $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$. M-RRT fails for 2 out of 50 runs (failed runs are not included in the table).

Algo	Solving time (s)					Number of nodes				
	Min	Q1	Med	Q3	Max	Min	Q1	Med	Q3	Max
M-RRT	0.22	2.67	6.20	15.95	45.46	16.0	83.8	185.5	419.8	1121.0
SPF	0.40	0.47	0.56	0.65	6.37	23.0	25.0	30.0	72.5	387.0

TABLE V: *UR3 swaps two spheres* results. With max 5000 iterations per run, M-RRT only succeeds in 48 out of 50 runs.

V. Conclusion

This paper proposes a general algorithm SPF to solve a wide range of prehensile manipulation planning problems representable by a constraint graph. Its unique way of propagating the constraints during RHS initialization tackles foliated grasp more efficiently than the M-RRT algorithm. The graph-search approach in SPF to generate transition sequences from the initial state to the goal state traverses large graphs more effectively than the random process deployed by M-RRT.

SPF can be improved by filtering out transition sequences with conflicting constraints. The optimality of the computed path is not the focus of the paper, but it is an important aspect. A strategy to update partial paths while planning or to post-process a non-optimal path can be considered. The ultimate goal is to create an algorithm that is practically fast and asymptotically optimal, for effective use in real applications.

REFERENCES

[1] J. Ota, "Rearrangement of multiple movable objects-integration of global and local planning methodology," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 2, pp. 1962–1967, IEEE, 2004.

[2] P. Lertkultanon and Q.-C. Pham, "A single-query manipulation planner," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 198–205, 2015.

[3] A. Krontiris and K. Bekris, "Dealing with difficult instances of object rearrangement," in *Robotics Science and Systems*, (Roma, Italy), 2015.

[4] R. Shome and K. E. Bekris, "Synchronized multi-arm rearrangement guided by mode graphs with capacity constraints," in *Workshop on the algorithmic foundations of robotics*, 2020.

[5] M. Gharbi, J. Cortés, , and T. Siméon, "Roadmap composition for multi-arm systems path planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Saint-Louis, USA), 2009.

[6] K. Harada, T. Tsuji, and J.-P. Laumond, "A manipulation motion planner for dual-arm industrial manipulators," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (Hongkong, China), pp. 928—934, 2014.

[7] A. Dobson and K. Bekris, "Planning representations and algorithms for prehensile multi-arm manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Hamburg, Germany), 2015.

[8] M. Stilman and J. Kuffner, "Planning among movable obstacles with artificial constraints," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1295–1307, 2008.

[9] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "An effective framework for path planning amidst movable obstacles," in *Algorithmic Foundation of Robotics VII*, pp. 87–102, Springer, 2008.

[10] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond, "Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1089–1103, 2013.

[11] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *International Journal of Robotics Research*, vol. 23, July 2004.

[12] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.

[13] V.-B. William and R. Nicholas, "Asymptotically optimal planning under piecewise-analytic constraints," in *Workshop on the algorithmic foundations of robotics*, 2016.

[14] J. Mirabel and F. Lamiraux, "Manipulation planning: addressing the crossed foliation issue," in *2017 IEEE International Conference on Robotics and Automation*, (Singapore, Singapore), May 2017.

[15] P. S. Schmitt, W. Neubauer, W. Feiten, K. M. Wurm, G. V. Wichert, and W. Burgard, "Optimal, sampling-based manipulation planning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3426–3432, IEEE, 2017.

[16] F. Lamiraux and J. Mirabel, "Prehensile Manipulation Planning: Modeling, Algorithms and Implementation," *IEEE Transactions on Robotics*, 2021.

[17] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, NY, USA: Springer, second ed., 2006.

[18] J. Mirabel, S. Tonneau, P. Fernbach, A.-K. Seppälä, M. Campana, N. Mansard, and F. Lamiraux, "HPP: A new software for constrained motion planning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 383–389, 2016.