2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Towards General Infeasibility Proofs in Motion Planning*

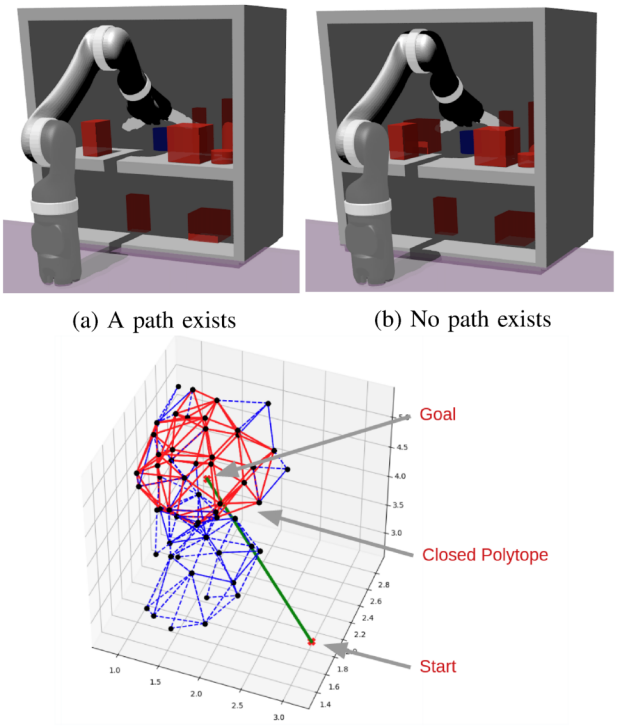
Sihui Li† Neil T. Dantam†

***Abstract*— We present a general approach for constructing proofs of motion planning infeasibility. Effective high-dimensional motion planners, such as sampling-based methods, are limited to probabilistic completeness, so when no plan exists, these planners either do not terminate or can only run until a timeout. We address this completeness challenge by augmenting a sampling-based planner with a method to create an infeasibility proof in conjunction with building the search tree. An infeasibility proof is a closed polytope that separates the start and goal into disconnected components of the free configuration space. We identify possible facets of the polytope via a nonlinear optimization procedure using sampled points in the non-free configuration space. We identify the set of facets forming the separating polytope via a linear constraint satisfaction problem. This proof construction is valid for general (i.e., non-Cartesian) configuration spaces. We demonstrate this approach on the low-dimensional Jaco manipulator and discuss engineering approaches to scale to higher dimensional spaces.**

I. Introduction

Motion planning is a well-studied topic with many applications. A continuing challenge is obtaining complete solutions to high-dimensional motion planning problems. Some complete or resolution complete methods have been developed using approaches such as cell decomposition [1], [2]; however, cell decomposition presents scalability challenges in higher dimensions. Widely used sampling based motion planning [3], [4] are only probabilistic complete, meaning if a plan exists they will find it in the limit (given enough time), but if no plan exists, they can run forever (or until a timeout). Numerous higher-level planning problems [5], [6], [7] would benefit from the ability to determine whether or not a motion planning problem is feasible—for example, can we reach a desired object or must we first remove an obstacle occluding the motion—and significant research effort has been devoted to working around this challenging issue [8], [9], [10], [11].

We propose a general method to construct motion planning infeasibility proofs based on sampling, optimization, and constraint-solving. We prove infeasibility by constructing a polytope whose facets are in the obstacle region of the configuration space and which separates the motion planning start and goal into separate components of the free configuration space. We generate candidate polytope facets in conjunction with a sampling-based planning approach by formulating and solving a nonlinear optimization problem for sampled, non-free points to fill the non-free space and separate the start and goal. We identify the set of facets that form the separating polytope by formulating and solving a linear constraint satisfaction problem that ensures the facets form a closed polytope and this polytope separates the start and goal. The result of this approach is either a motion plan, when one exists, or an infeasibily proof in the form of the separating polytope when no plan exists. We demonstrate our approach on the manipulator in Figure 1.



(a) A path exists (b) No path exists

(c) Separating configuration space polytope

Fig. 1: Examples of motion planning problems and an infeasibility proof. The goal is to reach the blue object at the back of the shelf. A path exists in (a) but not in (b). Our approach will generate the feasible plan for (a). For (b), we generate a proof of infeasibility in (c): a configuration space polytope that encloses the goal and excludes the start configurations.

Though this approach offers a general framework to construct infeasibility proofs for robot manipulators, challenges remain to practically scale to high-dimensional manipulators. Fortunately, the two key subroutines we employ—nonlinear optimization and linear constraint satisfaction—have both proven highly effective for solving combinatorially challenging planning problems [12], [13], [14], [15]. Thus, we anticipate that further engineering of the optimization and constraint formulation will enable this framework to provide practical infeasibility results in high-dimension. We discuss several of these engineering approaches in section V.

* This work supported in part by NSF grant IIS-1849348.
† Department of Computer Science, Colorado School of Mines, USA. li@mymail.mines.edu, ndantam@mines.edu.

II. Related work

Sampling-based motion planning has offered a practical and efficient approach to find motion plans in high-dimensional space [16], [4], [17], [18], [19]. However, these approaches are only probabilistically-complete and thus cannot identify cases when motion planning is infeasible. Our work directly addresses the challenge of proving motion planning infeasibility.

Some prior work has addressed special cases to prove path non-existence. Several approaches provide infeasibility for single, rigid bodies. [20] proves disconnection for a rigid body attempting to pass through gates. [21] proves path non-existence using alpha shapes in collision space. By constructing disconnected sets of 3-simplices in free configuration space from the exterior of the alpha shapes, if the start point and goal point are in different simplex sets, it proves no collision free path exists. In this work, scaling to higher dimensions depends on high dimensional α-shapes, which is still an open research question. [1] combines cell decomposition with a sampling based method to take advantage of both approaches. [2] identifies path non-existence by decomposing the configuration space into small rectangular cells and checking if a path exists entirely in collision-free cells. These methods are also only applied to 3-DOF robots because of the cost of decomposing the entire configuration space. [22] relates path non-existence for single, rigid objects in 2D or 3D workspace to the problem of caging. An approximation of the obstacle region is constructed to check whether an objects is caged, which proves path non-existence. [23] solves the motion planning problem where the workspace obstacles are polyhedral. They decompose the workspace into a set of polytopes and then setup a linear program to determine the feasibility of sub-plans when transitioning from one polytope to another.

Some other works focus on developing complete motion planning methods. [24] integrates PRMs (Probabilistic Roadmaps) with complete planning for discs moving in the plane. [25] constructs a road map based on star-shapedness which can terminate early when a feasible path is impossible. These methods decompose the search space, which may scale poorly. In contrast, our approach depends on filling only enough of the obstacle region to separate the start and the goal.

Other works have addressed problems related to motion planning infeasibility. [9] offers an asymptotically optimal algorithm to remove the minimum constraints (i.e., obstacles) to ensure that the motion planning problem is feasible. [26] provides a probability bounds on finding an optimal motion planning solution. Some approaches for task and motion planning (TMP) have developed approximate or heuristic estimates of motion planning feasibility [10], [8], [27]. Compared to these works, we directly address the challenge of proving that a particular motion planning problem is infeasible.

III. Algorithm

A. Problem Description

We address the problem of proving motion planning infeasibility. A motion planning problem consists of a configuration space $\mathcal{C}$ of dimension n, a start configuration $\boldsymbol{q}_{\text{start}}$ and a goal configuration $\boldsymbol{q}_{\text{goal}}$. The configuration space $\mathcal{C}$ is the union of the disjoint obstacle region $\mathcal{C}_{\text{obs}}$ and free space $\mathcal{C}_{\text{free}}$. Both $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are in $\mathcal{C}_{\text{free}}$. When a feasible plan exists, our approach will produce a plan σ such that $\sigma[0,1] \in \mathcal{C}_{\text{free}}$, $\sigma[0] = \boldsymbol{q}_{\text{start}}$, $\sigma[1] = \boldsymbol{q}_{\text{goal}}$. When there is no feasible plan, our approach will produce a proof of such.

B. Algorithm Overview

The algorithm generates a proof that the start and goal are in disconnected components of $\mathcal{C}_{\text{free}}$. The proof has two requirements: (1) a closed polytope whose facets are entirely within the $\mathcal{C}_{\text{obs}}$ and (2) $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are not both inside or both outside the polytope. In other words, the polytope separates the start and goal into separate components of $\mathcal{C}_{\text{free}}$.

We look for the separating polytope in conjunction with building a search tree in a sampling-based planner—specifically, in conjunction with RRT-Connect [17], though other sampling-based methods would also be applicable. The algorithm has two broad steps.

1) Generate a collision volume graph (CVG): Each time a configuration is sampled in the $\mathcal{C}_{\text{obs}}$, we generate a new collision ball and add it to the CVG. The collision balls represent a configuration space volume contained entirely in $\mathcal{C}_{\text{obs}}$. Within each ball, we identify n points, which provide a candidate facet for the separating polytope in the n dimensional space.
2) Construct the polytope: Given the CVG, we define a linear constraint satisfaction problem (LCSP) to identify the separating polytope. The constraints ensure that the polytope is closed and that it separates the start and goal. A solution to this LCSP proves that the start and goal are disconnected.

C. Construct the Collision Volume Graph (CVG)

Within the sampling-based planner, we use sampled configurations in the obstacle region $\mathcal{C}_{\text{obs}}$ to grow the CVG according to algorithm 1. We construct new balls based on two requirements: (1) fill as much of $\mathcal{C}_{\text{obs}}$ as possible and (2) intersect with the existing balls so that we can close the separating polytope. The idea of growing volumes in configuration space comes from [19].

Every vertex of the CVG is a ball. The vertex has four fields, the ball center, the radius, a set of n points in ball, and n sets of intersecting balls. The n points in the ball define a candidate facet of the separating polytope. The intersecting balls contain facets sharing a hyper-edge with the facet in the current ball. In n dimension, there are n different hyper-edges (choose $n-1$ from n) for each ball. If a hyper-edge of the current ball does not have an intersecting ball, we call this edge an *open edge*. If a ball has open edges, we call this ball an *open ball*.

Theorem 1: A polytope in $\mathbb{R}^n$ space is closed if and only if every facet of the polytope has an adjoining facet on each of its hyper-edges.

Proof: We prove by contradiction. If there exists one facet of a polytope with no adjoining facets on one of its edges, then the facet's inside and outside are connectable around this open edge. Since the inside and the outside of this polytope are connected. The polytope is not closed. ■

We use this requirement for closed polytopes to guide growth of the CVG in algorithm 1. We use a sample point to construct a collision ball only if it is in $\mathcal{C}_{\text{obs}}$ (line 1). If it is the first point sampled in the CVG, the algorithm finds an optimal start ball (line 2-5). The process of finding the optimal start impacts efficiency. A start ball that is too small will increase the number of balls and corresponding run time needed to form the polytope. We find an optimal start ball via a nonlinear optimization problem to maximize penetration depth (the distance to the nearest point on obstacle region boundary).

If the CVG is not empty, the algorithm first finds the ball in the CVG that is closest to the sampled point. Starting from the closest ball, we search for the first open ball and open edge (line 6-7). The new ball must include all points of the open edge. After finding an open ball/edge, there are three possible ways to generate new balls. The highest priority is to fill holes (line 8-11). A hole is a set of n points. Each of its $n-1$ points combinations is a hyper-edge of some balls, but the n points do not belong to any ball. If we find a hole containing all the edge points, we can create a new ball using the n points. After a hole ball is created, it needs to check all possible edge connections before returning.

If a hole cannot be found, we generate a new ball (line 12). To find the optimal ball center, we solve the following non-linear optimization problem:

$$\begin{aligned} \min_{c_{\text{new}}, r_{\text{new}}} \quad & \text{dist}(c_{\text{new}}, \text{line}(pe_{\text{mean}}, p_c)) - C * r_{\text{new}} \\ \text{s.t.} \quad & \text{dist}(pe_i, c_{\text{new}}) < r_{\text{new}}, i = 1, 2, \ldots, n-1 \\ & \text{penetration-dist}(c_{\text{new}}) > r_{\text{new}} \end{aligned}$$

where c_{new} and r_{new} are the new ball's center and radius, pe_i are the edge points of the open edge, pe_{mean} is the mean of edge points and p_c is the sampled point in collision. The first constraint ensures that all the edge points are in the new ball. The second constraint ensures the new ball radius is not larger than its penetration depth, thus the new ball is entirely in collision. The goal is to minimize the distance from the new ball center to the line connecting pe_{mean} and p_c, so that the new ball can grow onto the sampled direction as much as possible. At the same time, we want to maximize r_{new}. The constant C captures how much we want to prioritize growing larger balls.

After the new ball is generated, the ball contains $n-1$ points. There are two ways to add the last point. If there are existing points inside the new ball, use the closest one and check for all possible edge connections (line 13-18). Otherwise, generate a new point inside the ball (line 19). The new point generation uses the penetration vector (the vector connecting a point in collision and the closest point on the collision boundary). The new point needs to make the facet in the ball as perpendicular to the penetration vector as possible and being as far away to the existing points as possible. The final steps are to add edges between the new ball and the open ball, and add the new ball to the CVG (line 20-23).

Algorithm 1: Grow-Collision-Graph (p_c, G_c)

Result: A collision ball graph

1 **if** $p_c \notin \mathcal{C}_{\text{obs}}$ **then return**;
2 **if** G_c *is empty* **then**
3 $\quad v_{\text{new}}.c$, $v_{\text{new}}.r \leftarrow$ *find-optimal-start-ball*(p_c);
4 $\quad G_c$.*add-node*(v_{new});
5 $\quad$ **return**;
6 $v_{\text{closest}} \leftarrow$ *find-closest-ball*(G_c, p_c);
7 $v_{\text{open}} \leftarrow$ *find-open-ball*$(v_{\text{closest}}, G_c)$;
8 *fill-holes*(v_{open}, G_c);
9 **if** *a hole is filled* **then**
10 $\quad$ *check-edge-connection*(G_c);
11 $\quad$ return;
12 $v_{\text{new}}.c$, $v_{\text{new}}.r$ *find-new-ball*$(pc, v_{\text{open}}.c)$;
13 **for** v *in* G_c **do**
14 $\quad pin$.add(*intersection-points*(v, v_{new}));
15 **if** $\|pin\| \geq 1$ **then**
16 $\quad p_{\text{new}} \leftarrow$ *find-closest-intersection-point*(pin, p_c);
17 $\quad$ *check-edge-connection*(G_c);
18 **else**
19 $\quad p_{\text{new}} \leftarrow$ *generate-new-points*(v_{new})
20 v_{new}.*add-point*(p_{new});
21 v_{new}.*add-edge*(v_{close});
22 v_{close}.*add-edge*(v_{new});
23 G_c.*add-node*(v_{new});

D. Configuration-space penetration depth

The collision balls represent configuration-space collision volumes; however, existing collision detection approaches can only help us identify collision information such as penetration depth in the Cartesian workspace [28]. To generate the collision balls, we need penetrations depths and vectors in configuration space. We define the workspace to configuration space collision relationship as a nonlinear optimization problem to minimize the distance between the point in collision and the collision boundary point,

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \text{dist}(\boldsymbol{q}_c, \boldsymbol{q}) \\ \text{s.t.} \quad & \text{ssd}(\vec{x}_m(\boldsymbol{q}) - \vec{x}_o) = 0 \,, \end{aligned} \tag{1}$$

where $\boldsymbol{q}_c$ is the configuration in collision, $\boldsymbol{q}$ is the collision boundary configuration we want to find, $\vec{x}_m$ is Cartesian point of maximum penetration on the robot, $\vec{x}_0$ is the Cartesian boundary point on the obstacle. The optimization goal is the shortest configuration space distance, subject to the sum of squared distance (ssd) between the point on the

robot and the point on the obstacle being zero. We use a local search to solve this optimization problem.

E. Constructing a closed polytope

The first step of constructing the separating polytope is ensuring the selected facets form a closed polytope. According to Theorem 1, each hyper-edge of each facet must be adjoined by another facet. For the CVG, we can thus say that for each ball b_i, the polytope must also contain one ball from each of b_i's n sets of intersection balls,

$$b_i \implies \begin{cases} b^i_{11} + b^i_{12} + \ldots + b^i_{1n^i_1} = 1 \\ \ldots \\ b^i_{j1} + b^i_{j2} + \ldots + b^i_{jn^i_j} = 1 \\ \ldots \\ b^i_{n1} + b^i_{n2} + \ldots + b^i_{nn^i_n} = 1 \end{cases} , \tag{2}$$

where b_i indicates a ball is used to form the polytope and b^i_{jk} indicates that a ball intersecting b_i is used to form the polytope. b^i_{j1}, b^i_{j2}, ..., $b^i_{jn^i_j}$ are the jth intersection set of ith ball. n^i_j is the number of balls in the jth intersection set.

We rewrite (2) as linear inequality constraints. For each intersection set we have,

$$(1 - b_i) + (b^i_{j1} + b^i_{j2} + \ldots + b^i_{jn^i_j}) > 0 \tag{3}$$

and

$$b^i_{j1} + b^i_{j2} + \ldots + b^i_{jn^i_j} \leq 1 + (1 - b_i)m, \tag{4}$$

where m is the number of balls.

Equation (3) ensures that if b_i is chosen, at least one ball must be chosen from each of its intersection set. Equation (4) ensures that if b_i is chosen, then the total number of balls is less than or equal to 1. These two inequalities together ensure one and only one ball from each intersection set. At the same time, if b_i is 0, there is no requirement to include balls from the intersection sets.

In addition, we need at least $n+1$ balls to form a polytope, which gives us:

$$b_1 + b_2 + \ldots + b_m \geq n + 1 \ . \tag{5}$$

There are $2n$ inequalities for each ball. The total number of inequalities is $2nm$.

F. Separating the start and goal

Next, we ensure that the polytope separates the start and goal configurations by applying the Ray casting algorithm [29]. We find the line segment between the start and goal and count the intersections between the line segment and the polytope facets. If there is an odd number of intersections, one point must be inside the polytope and the other outside. Thus, we know that the polytope separates the start and goal.

Each polytope facet lies in a hyperplane in $\mathbb{R}^n$,

$$a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = 1 \ . \tag{6}$$

The point set in a ball b_i is p^i_1, $p^i_2, \ldots, p^i_n$, and each point is an n vector, $p^i_j = [p^i_{j1}, p^i_{j2}, \ldots, p^i_{jn}]^T$. We find the hyperplane from the points:

$$(a^i_1 p^i_{j1} + a^i_2 p^i_{j2} + \ldots + a^i_n p^i_{jn}) = 1, j = 1, \ldots, n \ , \tag{7}$$

where $a^i_1, \ldots, a^i_n$ are the parameters for the hyperplane in the ith ball.

In n dimension, the line connecting $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ is,

$$\frac{x_1 - \boldsymbol{q}^1_{\text{goal}}}{\boldsymbol{q}^1_{\text{start}} - \boldsymbol{q}^1_{\text{goal}}} = \frac{x_2 - \boldsymbol{q}^2_{\text{goal}}}{\boldsymbol{q}^2_{\text{start}} - \boldsymbol{q}^2_{\text{goal}}} = \ldots = \frac{x_n - \boldsymbol{q}^n_{\text{goal}}}{\boldsymbol{q}^n_{\text{start}} - \boldsymbol{q}^n_{\text{goal}}} \ , \tag{8}$$

with $\boldsymbol{q}_{\text{start}} = [\boldsymbol{q}^1_{\text{start}}, \boldsymbol{q}^2_{\text{start}}, \ldots, \boldsymbol{q}^n_{\text{start}}]^T$ and $\boldsymbol{q}_{\text{goal}} = [\boldsymbol{q}^1_{\text{goal}}, \boldsymbol{q}^2_{\text{goal}}, \ldots, \boldsymbol{q}^n_{\text{goal}}]^T$. The line definition gives us $n-1$ equations, together with the hyperplane equation, we have a total of n equations to solve for the intersection point $x_i = [x^1_i, \ldots, x^n_i]$ between the line and the hyperplane in the ith ball.

First, we determine whether the intersection between the line and hyperplane lies on the segment between $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$:

$$\begin{aligned} x_i - \boldsymbol{q}_{\text{start}} &= l_i \frac{\boldsymbol{q}_{\text{goal}} - \boldsymbol{q}_{\text{start}}}{\|\boldsymbol{q}_{\text{goal}} - \boldsymbol{q}_{\text{start}}\|} \\ l_i &< \|\boldsymbol{q}_{\text{goal}} - \boldsymbol{q}_{\text{start}}\| \\ l_i &> 0 \ . \end{aligned} \tag{9}$$

Second, we ensure the intersection point lies within the facet bounds, i.e., which is the area enclosed by points used to define the hyperplane (i.e., the ball's point set becomes the facet's vertices). Each hyperplane is formed by n points. On the hyperplane, the points can be considered as points in $n-1$ dimensional space. By reducing one dimension, the problem of a point inside the hyperplane area in n dimensional space can be converted to the problem of a point inside a polytope in $n-1$ dimensional space. Since we have n points on the hyperplane, we have n points in the $n-1$ dimensional space, which form a simplex. To determine if a point is inside a simplex, we apply Carathéodory's theorem [30] and calculate Barycentric coordinates. A point p in $n-1$ dimension is inside a n-1-simplex formed by n points $[v_1, v_2, \ldots, v_n]$ if all of its Barycentric coordinates are positive and the sum of the coordinates is less than or equal to 1. We calculate the Barycentric coordinates can by,

$$\lambda = T^{-1}(p - v_n) \ , \tag{10}$$

where T is ,

$$T = [v_1 - v_n, \ldots, v_{n-1} - v_n]^T \ , \tag{11}$$

and the Barycentric coordinates are,

$$\begin{aligned} \lambda &= (\lambda_1, \ldots, \lambda_{n-1}), \\ \lambda_n &= 1 - \sum_{i=1}^{n-1} \lambda_i \ . \end{aligned} \tag{12}$$

Point p is inside the simplex if the Barycentric coordinates satisfy,

$$\lambda_i \geq 0, i = 1, \ldots, n \ . \tag{13}$$

Applying this result to our case, we have point set p_1^i, $p_2^i, \ldots, p_n^i$ in n dimension for ball b_i. We use the Gram–Schmidt procedure on $[p_1^i - p_n^i, p_2^i - p_n^i, \ldots, p_{n-1}^i - p_n^i]$ to give a $n-1$ dimension orthogonal coordinate system. We represent the intersection point in this new coordinate, then calculate the Barycentric coordinates.

If both (9) and (13) are satisfied, then the ball's facet is intersected by the line between the start and goal. We call such balls *essential balls*, denoted by E_i. We can always check whether a ball is essential when we construct the CVG, since we always know balls' point sets.

Given a polytope, we must also ensure that there are an odd numbers of intersections according to the Ray casting algorithm, i.e., an odd number of cases where both b_i and E_i are 1. This requirement yields the constraint,

$$b_1 E_1 + b_2 E_2 + \ldots + b_m E_m = 2r + 1 \,, \tag{14}$$

where r is a non-negative integer. Combining (14) with (3), (4), and (5) yields the complete LCSP. Since E_i is known for each ball in the CVG, the variables we must solve in this problem are all the b_is and r. A solution to this LCSP defines the separating polytope and provides a proof that no feasible motion plan exists.

G. Two Simple Examples

We illustrate the approach using two simple examples in 2D and 3D.

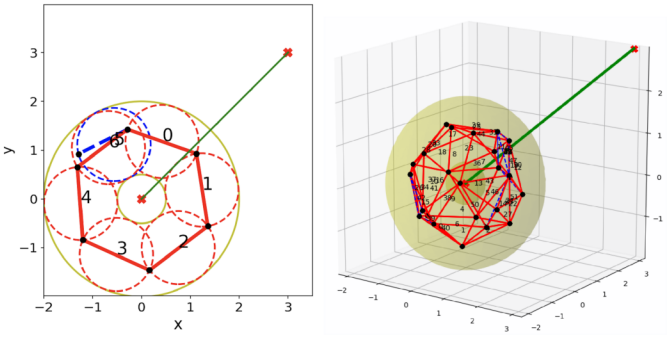
Figure 2a shows a 2D disconnection proof. The green line connects the start and goal point. The space in between the yellow circles is in collision. In 2D, the balls are circles, facets are lines inside a ball, and hyper-edges are points. Ball 0 is first added to the CVG. Before any more balls are added, both hyper-edges (points in 2D) of ball 0 are open. Ball 1 is grown onto one side of ball 0's open edges. Each time a point is added to the CVG, we calculate the ball's essential property E. As can be seen Figure 2a, only one ball is essential (ball 0).

Balls 2-4 grow onto the open edges, and new points are generated until after we generate ball 6. Ball 6 has one of ball 0's point inside it, so we are using this existing point instead of generating a new point. After checking all edge connections, ball 0 and ball 6 are connected. At this point, if we check the LCSP for this CVG, a closed polytope with only one essential facet (ball 0) exists, which satisfies (14).

Figure 2b shows a similar infeasibility proof for the 3D case. In this example, we define $\mathcal{C}_{\text{obs}}$ to be the region between two balls of radius 0.5 and 2.0. In 3D, the facets are triangles and the hyper-edges are the triangles' edges. Again, we grow the facets onto the open hyper-edges one by one. A total of 51 balls are generated in this case. The red triangles with solid edges are the triangles decided by the LCSP to form the polytope. The blue triangles with dashed edges are not included in the polytope. The collision balls are not shown in Figure 2b for clarity.

IV. Experiments

We evaluate our approach for the Jaco manipulator on the scenes similar to Figure 1. In our experiments, we adapt RRT-Connect [17] in OMPL [18] to simultaneously grow a search tree and CVG. We solve the nonlinear optimization problems using sequential least-squares quadratic programming (SLSQP) [31], we find workspace collisions and distances using the Flexible Collision Library [28], we solve the LCSP using Z3 [32], and we model robot kinematics using Amino [12], [33].



(a) A 2D example. (b) A 3D example.

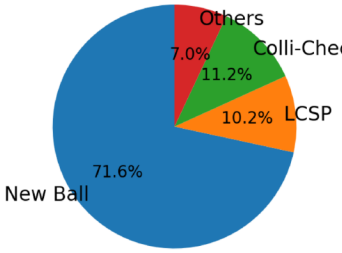
Fig. 2: Example separating polytopes in 2D and 3D. Red markers are the start point and the goal point. Yellow circles/balls are the obstacle region.The red/solid facets form a closed polytope. The blue/dashed facets is not included in the polytope. (a) shows a 2D infeasibility proof. (b) shows a 3D infeasibility proof.

We evaluate overall runtime and overhead of our approach. We test six different scenes: three scenes where a path exists and three scenes where no path exists. For each scene, we run 12 trials. For the scenes with feasible paths, we compare the overhead introduced by attempting to construct the infeasibility proof with an unmodified RRT-Connect. When no path exists, a traditional sampling-based planner runs until a timeout, whereas, using our infeasibility proof construction, the planner is able to terminate when it constructs the infeasibility proof.

Figure 3 shows profiling results of overall runtime to construct the infeasibility proof. The three functions requiring the majority of the runtime are the LCSP solver, optimization to find the new balls, and the collision checking. The process to find penetration distance, which is called in each iteration of the optimization process to find new balls, took over 99% of the optimization time. We discuss approaches to address these bottlenecks in section V.

For the scenes where a path exists, we compared the overhead introduced by our approach to the unmodified RRT-Connect. RRT-Connect required an average runtime of 13.65s, whereas adding the infeasibility proof construction required an average of 284.06s due to the additional computation to construct the CVG and search for the separating polytope.

During these manipulator experiments, we also observe some properties of the configuration space. Configuration space penetration vectors may not change continuously. The penetration vectors can change a lot given a small change in the collision point's coordinate. This causes the number of balls in the real obstacle region to be considerably larger than



	Mean(s)	STD(s)
Total Time	457.5	396.24
LCSP	46.62	41.77
New Ball	327.59	188.54
Colli-Check	51.24	145.05
Get Pene-Dist	326.79	172.96

Fig. 3: Runtime distribution (left). Mean and STD (right).

the Cartesian 3D space in the simple examples (Figure 2b). The average number of balls for the manipulator is 174, compared to around 50 for the simple 3D case. We must also place a maximum limit on the balls radius to ensure small regions can be covered. Figure 1c shows the configuration space facets in one of the scenes where a path does not exist.

To compare with the previous method [21], [1], [2], which also performed experiments on configuration spaces (either for manipulators or single objects), one advantage of our method is that the sampling does not need to fill the entire $\mathcal{C}_{\text{obs}}$ as long as we can construct the closed polytope. In general, this means we do not need to process a large number of sampling points in $\mathcal{C}_{\text{obs}}$ that are not related to the infeasibility problem.

V. Discussion and Future Work

This algorithm and experiments offer initial results towards general case proofs of motion planning infeasibility. The presented formulation supports the non-Cartesian configuration spaces of manipulators and is generalizable to higher dimensions. A key challenge lies in engineering the sampling, search, and constraints to scale effectively in higher dimensions. We discuss approaches to address computational bottlenecks and scale this work.

First, optimizing collisions balls is a current bottleneck. To produce each collision ball, we must compute penetration depths—an expensive operation—multiple times. We will investigate approximate solutions and caching of collision results to improve this process. Furthermore, generating collisions balls is highly parallelizable since we may concurrently optimize the balls for multiple open hyper-edges.

The local search procedure to calculate configuration-space penetration can also be improved. In (1), given that we can obtain the correct points of maximum penetration $\vec{x}_m$ and collision boundary $\vec{x}_o$, the gradient of the constraint function would be $\nabla \operatorname{ssd} \mathbf{J}_m$, where $\nabla \operatorname{ssd} = 2(\vec{x}_m - \vec{x}_o)^T$, and $\mathbf{J}_m$ is the manipulator Jacobian. Gradient-based methods to solve (1) are potentially faster than the local search method we implemented.

Second, we may reduce the time to solve the LCSP by leveraging incremental constraint solving. Each time we add a new collision ball, we construct and solve an entirely new LCSP. However, these LCSPs are closely related. Incremental constraint solving approaches rapidly solve related problems by reusing prior search effort [32]. Our previous work has demonstrated the value of incremental constraint solving for planning problems [8]. Developing a process to incrementally add and solve constraints for the LCSP may be especially helpful in higher dimension where many more collision balls may be generated.

Finally, the search for collision balls is a key step that effects scalability. Generating larger numbers of collision balls results in a larger LCSP, which is slower to solve. Moreover, the generation process also effects convergence to a closed polytope. In the current approach, optimizing the facets to be perpendicular to the penetration vector offers a heuristic to help ensure that we can add further collision balls that neighbor the current facet. Meanwhile, the step of filling holes during the generation process improves convergence because filling a hole usually closes more open edges compared to generating a new ball. For the same reason, the step of filling holes also reduces the number of balls generated. We will investigate further heuristics to improve convergence to a closed polytope with a small number of ball. It is possible that with good heuristics, we will not need to setup the LCSP problem.

VI. Conclusion

We have presented a general method for motion planning infeasibility proofs and demonstrated the approach on a low-dimensional manipulator. Our approach progressively grows volumes in the obstacle region via sampling and optimization and then attempts to find a closed polytope separating the start and goal via constraint solving. This method is applicable to manipulator configuration spaces widely addressed by sampling-based planners. To practically apply this approach to high-dimensional path non-existence, we will further engineer the generation of collision volumes and the constraint solving process.

References

[1] L. Zhang, Y. J. Kim, and D. Manocha, "A hybrid approach for complete motion planning," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2007, pp. 7–14.

[2] ——, "A simple path non-existence algorithm using c-obstacle query," in *Algorithmic Foundation of Robotics VII*. Springer, 2008, pp. 269–284.

[3] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Department, Iowa State University, Tech. Rep. TR 98-11, 1998.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

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

[6] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *International Journal of Robotics Research (IJRR)*, vol. 28, no. 1, pp. 104–126, 2009.

[7] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 549–565, 1998.

[8] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[9] K. Hauser, "The minimum constraint removal problem with three robotics applications," *International Journal of Robotics Research*, vol. 33, no. 1, pp. 5–17, 2014.

[10] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *International Journal of Robotics Research*, vol. 35, no. 8, pp. 890–927, 2016.

[11] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics (IJHR)*, vol. 2, no. 04, pp. 479–503, 2005.

[12] N. T. Dantam, S. Chaudhuri, and L. E. Kavraki, "The task motion kit," *Robotics and Automation Magazine*, vol. 25, no. 3, pp. 61–70, 2018.

[13] H. Kautz and B. Selman, "Blackbox: A new approach to the application of theorem proving to problem solving," in *AIPS98 Workshop on Planning as Combinatorial Search*, vol. 58260, 1998, pp. 58–60.

[14] J. Rintanen, "Engineering efficient planners with SAT," in *Eu. Conference on Artificial Intelligence (ECAI)*, 2012, pp. 684–689.

[15] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[16] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[17] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Intl. Conference on Robotics and Automation*, vol. 2. IEEE, 2000, pp. 995–1001.

[18] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *Robotics & Automation Magazine (RAM)*, vol. 19, no. 4, pp. 72–82, 2012.

[19] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space," *arXiv:1109.3145v1 [cs.RO]*, 2011.

[20] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen, "Disconnection proofs for motion planning," in *International Conference on Robotics and Automation*, vol. 2. IEEE, 2001, pp. 1765–1772.

[21] Z. McCarthy, T. Bretl, and S. Hutchinson, "Proving path non-existence using sampling and alpha shapes," in *International Conference on Robotics and Automation*. IEEE, 2012, pp. 2563–2569.

[22] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic, "Caging and path non-existence: a deterministic sampling-based verification algorithm," in *Robotics Research*. Springer, 2020, pp. 589–604.

[23] Y. Shoukry, P. Nuzzo, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Scalable lazy smt-based motion planning," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 6683–6688.

[24] S. Hirsch and D. Halperin, "Hybrid motion planning: Coordinating two discs moving among polygonal obstacles in the plane," in *Algorithmic Foundations of Robotics V*. Springer, 2004, pp. 239–255.

[25] G. Varadhan and D. Manocha, "Star-shaped roadmaps-a deterministic sampling approach for complete motion planning." in *Robotics: Science and Systems*, vol. 173. Citeseer, 2005.

[26] A. Dobson, G. V. Moustakides, and K. E. Bekris, "Geometric probability results for bounding path quality in sampling-based roadmaps after finite computation," in *International Conference on Robotics and Automation*. IEEE, 2015, pp. 4180–4186.

[27] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE robotics and automation letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

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

[29] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, "A characterization of ten hidden-surface algorithms," *ACM Computing Surveys (CSUR)*, vol. 6, no. 1, pp. 1–55, 1974.

[30] H. G. Eggleston, *Helly's Theorem and its Applications*, ser. Cambridge Tracts in Mathematics. Cambridge University Press, 1958, p. 33–44.

[31] D. Kraft, "A software package for sequential quadratic programming," Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, Tech. Rep. DFVLR-FB 88-28, July 1988.

[32] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[33] N. T. Dantam, "Practical exponential coordinates using implicit dual quaternions," in *Workshop on the Algorithmic Foundations of Robotics*, 2018.