A Simple Path Non-Existence Algorithm for low DOF robots

http://gamma.cs.unc.edu/nopath

Liangjun Zhang[1], Young J. Kim[2], and Dinesh Manocha[1]

[1] Dept. of Computer Science, University of North Carolina at Chapel Hill, U.S.A.
zlj,dm@cs.unc.edu
[2] Dept. of Computer Science and Engineering, Ewha Womans University, Korea,
kimy@ewha.ac.kr

Technical reports TR06-006, Dept. of Compute Science, UNC Chapel Hill

Summary. We present a simple algorithm to check for path non-existence for a robot among static obstacles. Our algorithm is based on adaptive cell decomposition of C-space. We use two basic queries: C-free query, which checks whether a cell in C-space lies entirely in the free space, and C-obstacle query, which checks whether a cell lies entirely inside the C-obstacle space. Our approach reduces the path non-existence problem to checking whether there exists a path through cells that do not belong to the C-obstacle space. We describe simple and efficient algorithms to perform C-free and C-obstacle queries using separation and *generalized penetration distance* computations. Our algorithm is simple to implement and we demonstrate its performance on 3-DOF robots.

1 Introduction

Motion planning is a fundamental problem in robotics. The goal is to compute a collision-free path between two configurations of a given robot. This problem is extensively studied in the field for more than three decades. At a broad level, prior algorithms for motion planning can be classified into roadmap methods, exact cell decomposition, approximate cell decomposition, potential field methods and randomized sampling-based methods [14]. In particular, planning algorithms such as the roadmap methods and exact cell decomposition are referred to as complete motion planning algorithms. These approaches can compute a collision-free path if one exists; otherwise they report path non-existence between the two configurations. However, these methods are known to have a high theoretical complexity and their practical implementations are limited to planar robots or convex polytopes or special shapes such as spheres or ladders.

Practical algorithms for motion planning are based on approximate cell decomposition, potential field computation or sampling-based algorithms. The

approximate cell decomposition algorithms subdivide the configuration space into cells and can be made *resolution complete* based on a suitable choice of parameters. However, prior approaches can result in a very high number of subdivisions and can be overly conservative. Moreover, cell decomposition requires enumeration of contact surfaces and for a robot with high geometric complexity, generating and enumerating contact surfaces can be complicated and time consuming [8, 15].

On the contrary, randomized sampling methods such as probabilistic roadmap planners (PRMs) are relatively simple to implement and work quite well in practice [11]. The strength of PRMs lies in their simplicity and they can be easily applied to general robots with high DOFs. However, PRMs have two major issues: path non-existence (i.e. no passage) and narrow passages. If there is no collision-free path, PRM algorithms do not terminate. Moreover, it is hard to distinguish whether such situations arise due to path non-existence or due to narrow passages and poor sampling.

Main Results

In this paper, we present a simple cell decomposition based algorithm that is capable of reporting that there exists no path from initial to goal configurations of a robot. Our resulting algorithm is a complete motion planning algorithm for a rigid robot with translational and rotational DOFs. Furthermore, our approach also extends to articulated robots. We subdivide the configuration space into *empty*, $full$ and *mixed* cells. Unlike prior cell decomposition algorithms, we subdivide only the region that is critical for path existence. Furthermore, we use efficient algorithms to classify a given cell as $full$ by checking whether it lies in C-obstacle space. As a result, our approach generates fewer cells and subdivisions as compared to prior approaches.

Our algorithm uses two kind of queries: $C - free$ query and $C - obstacle$ query. Given a region (typically, a rectangloid) in the configuration space (C-space), these queries check whether a region belongs completely to the free space or the C-obstacle space. We efficiently perform these queries in the workspace by computing the Euclidean distance and *generalized penetration depth*. Based on these queries, our algorithm subdivides the configuration space into *empty*, $full$ and *mixed* cells, where *empty* and $full$ cells are classified based on C-free and C-obstacle queries, respectively. In order to check for path non-existence, we search through a sequence of adjacent *empty* or *mixed* cells to find the goal configuration. The non-existence of such a sequence is a sufficient condition for path non-existence between the start and the goal configuration. We have implemented our algorithm and we highlight its performance on 3-DOF robots.

Organization

The rest of the paper is organized as follows. In Section 2, we briefly survey work related work on motion planning. We give an overview of our algorithm in Section 3 and present our cell labelling algorithm in Section 4. We describe our implementation and highlight a few limitations of our approach in Section 5.

2 Previous Work

Motion planning has been extensively studied in the field for more than three decades. An excellent survey of this topic is available in [14]. In this section, we briefly review prior algorithms for exact and approximate motion planning.

2.1 Exact Motion Planning

The complete motion planning algorithms compute a collision-free path if one exists; otherwise they report path non-existence. These include criticality-based algorithms such as exact free-space computation for a class of robots [16, 12, 3, 9], roadmap methods [7], and exact cell decomposition methods [19]. The exact cell decomposition methods require an exact description of the configuration space consisting of C-free and C-obstacle regions. The boundary between C-free and C-obstacle regions is described by a set of contact surfaces, each surface being the locus of configurations of a robot at which a specific boundary feature of the robot is in contact with a boundary feature of the obstacles. The exact cell decomposition approaches partition the C-free space into a collection of simpler geometric regions and compute a connectivity graph representing the adjacency between the regions.

In theory, these methods are quite general. However, in practice, it is quite challenging to implement them and no good implementations are known for general robots. In particular, computing and enumerating contact surface in the configuration space becomes hard. For a 6-DOF robot with k polyhedral boundary features, each contact surface is a 5-dimensional manifold, whose boundary lies in the 6-dimensional configuration space and their combinatorial complexity can be as high as $O(k^2)$ [8]. As a result, many variants have been proposed to deal with special cases of motion planning problems; these include planar objects with 3-DOF, convex polytopes in 3D, polyhedral objects in 3D with translational DOFs, and specially shaped objects such as ladders, discs, and balls [14].

2.2 Approximate Cell Decomposition and Sampling-based Approaches

A number of algorithms based on approximate cell decomposition have been proposed [5, 8, 26]. These methods partition the C-space into a collection of cells. They classify the cells into three types: *empty* cells that lie completely in the free space, *full* cells that are completely within C-obstacle space, and *mixed* cells that correspond to the rest. Unlike exact cell decomposition, the cells used in approximate cell decomposition algorithms have a simpler, rectangloid shape, and the *empty* cells provide a conservative approximation of the free space. The planner searches through the empty cells to find a path. Moreover, approximate cell decomposition methods are *resolution complete*; i.e., they can find a path if one exists provided the resolution parameters are selected small enough [14]. In practice, approximate cell decomposition methods have been used for low DOF robots.

The probabilistic roadmap method (PRM) [11] is perhaps the most widely used path planning algorithm and used for different applications. It is relatively simple to implement and has been successfully applied to high DOF robots. Since PRM-based algorithms sample the free space randomly, they may fail to find paths, especially those passing through narrow passages. A number of extensions have been proposed to improve the sampling in terms of handling narrow passages [2, 10, 17] or use visibility-based techniques [21]. In order to choose the best planner among these variants depending on problem features, a C-space subdivision method called 'meta planner' has been introduced [1]. An approach for creating tree-like nodes in probabilistic roadmaps has been introduced and this approach can be parallelized and applied to high DOF robots [18]. All these methods are *probabilistically complete*.

Recently, a deterministic sampling approach called Star-shaped roadmaps has been proposed [23]. The free space is partitioned into star-shaped regions and connectors between star-shaped regions are computed for inter-region connectivity. This algorithm is complete as long as there are no tangential contacts in the boundary of the free space. Since Star-shaped roadmaps compute global connectivity of free space, the number of regions can increase exponentially as a function of DOFs. Moreover, this approach also suffers from the complexity of contact surface enumeration.

2.3 Path Non-Existence

Exact planning approaches such as exact cell decomposition and roadmap computation can check for path non-existence. However, these methods are not practical due to their theoretical complexity and implementation difficulty. In general, a popular planning method such as PRM cannot deterministically guarantee the path non-existence as it is only probabilistically complete. An effort has been made to address the issue of path non-existence in PRM [4]. The authors have proposed a *disconnection prover*, probabilistically showing that the motion planning problem has no solution. However, this approach is restricted to a special problem of finding a path through a planar section. The deterministic sampling approach such as the star-shaped roadmaps [23] is a complete approach but it may be overly conservative and generate a high number of samples.

3 Overview

In this section, we give an overview of our algorithm. As compared to prior cell decomposition approaches, our algorithm has two distinct features. During cell decomposition, we use a reliable algorithm to label each cell as *empty*, *full* or *mixed*. Moreover, prior cell decomposition algorithms enumerate the contact surfaces, whereas our cell labelling algorithm only relies on two specific distance metric computations: separation distance and generalized penetration depth computation. Moreover, our algorithm builds a *connectivity graph* with the *empty* and *mixed* cells. We check for path non-existence between

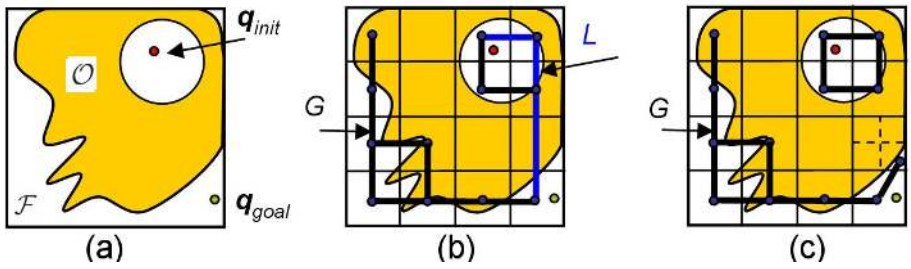

Fig. 1. *Path non-existence between* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$*. (b): A connectivity graph* G *is built. The path* L*, which connects the cells including* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$*, is computed from* G*. Any mixed cell along* L *is further subdivided. (c): In the new connectivity graph, the cell containing* $\mathbf{q}_{init}$ *and the cell containing* $\mathbf{q}_{goal}$ *are disconnected and therefore, there is no collision-free path between* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$*.*

the initial and goal configurations by performing a search in the *connectivity graph*.

Algorithm 1 Path non-existence Algorithm
Input: Robot $\mathcal{A}$, obstacle $\mathcal{B}$, initial and goal configurations $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$.
Output: Report whether there exists no path from $\mathbf{q}_{init}$ to $\mathbf{q}_{goal}$.

```
1: Q := Bounding box of configuration space
2: repeat
3:    for each cell C in Q do
4:       Label C with full, empty or mixed
5:    end for
6:    G := Connectivity graph of all full cells
7:    G_e := The connectivity subgraph of G for all empty cells
8:    C_init := The cell containing q_init
9:    C_goal := The cell containing q_goal
10:   if FindPath(G_e, C_init, C_goal) then
11:      Report the found path
12:      return 'Exists a path from q_init and q_goal'.
13:   else if L := FindPath(G, C_init, C_goal) then
14:      return 'Exists no path from q_init and q_goal'.
15:   else
16:      Q := φ
17:      for each mixed cell C along L do
18:         Q += Subdivided cells from C
19:      end for
20:   end if
21: until true
```

3.1 Notation

We use the symbol $\mathcal{A}$ to denote a robot and $\mathcal{B}$ to represent the collection of all fixed obstacles. Let $\mathcal{C}$ denote the configuration space of the robot. $\mathcal{F}$ and $\mathcal{O} = \mathcal{C} \backslash \mathcal{F}$ represent free space and configuration obstacle space (or C-obstacle), respectively. A cell C in n-dimensional C-space is defined as a Cartesian product of real intervals:

$$C = [x_1', x_1''] \times [x_2', x_2''] \cdots \times [x_n', x_n''].$$

We denote $\mathcal{A}(\mathbf{q})$ as a placement of a robot $\mathcal{A}$ at configuration $\mathbf{q}$. Let $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ represent the initial and goal configurations of the robot, respectively. A line segment in C-space connecting configurations $\mathbf{q}_a$ and $\mathbf{q}_b$ is represented as $\pi_{\mathbf{q}_a,\mathbf{q}_b}$.

Let $l(\mathbf{t}), t \in [0,1]$ be an arbitrary curve in the C-space. If the configuration of $\mathcal{A}$ is set as $l(\mathbf{t})$, when t changes from 0 to 1, any point $\mathbf{p}$ on $\mathcal{A}$ traces a distinct curve in the workspace. We call such an event as moving a robot along l. Let $\mu(\mathbf{p}, l)$ be defined as the length of a curve traced by the point $\mathbf{p}$ when A moves along l.

3.2 Cell Decomposition and Labelling

Our approach to check for path non-existence is based on cell decomposition. The configuration space $\mathcal{C}$ is spatially subdivided into cells at successive levels of the subdivision. The cells are classified as *empty* or *full* depending on whether they lie entirely inside the free space $\mathcal{F}$, or entirely inside the C-obstacle $\mathcal{O}$. If they are neither *empty* nor *full*, they are labelled as *mixed*. Fig.'s 1 and 2 illustrate that cells are labelled with different types. We present our cell labelling algorithm in Section 4.

3.3 Connectivity Graphs

For each level of subdivision, the connectivity graph G is built to represent the adjacency relationship among *empty* and *mixed* cells. Formally, the connectivity graph [26, 27, 14] associated with a decomposition $\mathcal{D}$ of $\mathcal{C}$ is an undirected graph, where:

- The vertices in G are the *empty* and *mixed* cells in $\mathcal{D}$.
- Two vertices in G are connected by an edge if and only if the corresponding cells are adjacent.

Intuitively, G captures the connectivity of both the identified free space, which is covered by the *empty* cells, and the 'uncertain' region, which is represented by the *mixed* cells.

In order to check for path non-existence, our algorithm first locates the cells C_{init} and C_{goal}, which contain $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, respectively. Next, the algorithm searches G to find a path L, a sequence of adjacent *empty* and *mixed* cells connecting C_{init} and C_{goal} (Fig. 1). If no such path is found, it is a sufficient criterion to guarantee that there is no collision-free path that

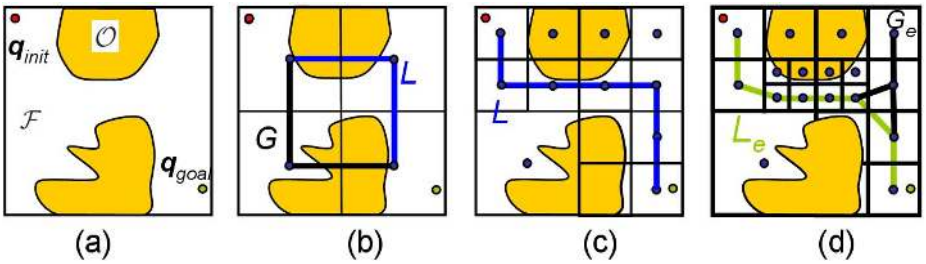

Fig. 2. *Planning a collision-free path from* $\mathbf{q}_{init}$ *to* $\mathbf{q}_{goal}$*. The shaded regions denote C-obstacle space. In (b), after the first level of subdivision, a connectivity graph G, which captures the connectivity of the boundary of the free space, is constructed. (b), (c): The path L is computed from the connectivity graph and is used to guide which 'uncertain' regions have higher priorities in terms of further subdivision. (d): A collision-free path* L_e *is computed by searching the subgraph* G_e *for all empty cells.*

connects $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$. Therefore, our algorithm can safely report that $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ are disconnected.

There are various known heuristics that can prioritize the search on the connectivity graph G. We use the shortest path algorithm to search for a path connecting C_{init} and C_{goal} on G. We also assign each edge a different weight, where the edge associated with two *empty* cells has the smallest weight (0 in our implementation) and the one with two *mixed* cells has the largest weight.

Our algorithm terminates if a collision-free path is computed (see Fig. 2) or we can prove path non-existence. For this purpose, a subgraph G_e of G is constructed. G_e represents the adjacency relation among all *empty* cells. Intuitively, G_e represents the connectivity of a part of free space that has been identified till the current level of subdivision. If there is a path in G_e connecting C_{init} and C_{goal}, a collision-free path can be easily extracted and optimized [26].

3.4 Guided Subdivision

When a path L is reported after searching the connectivity graph G, it is not clear whether $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ are disconnected. If so, we need to further explore the 'uncertain' regions - the union of *mixed* cells, to acquire more information about their connectivity. A simple approach is to apply another level of subdivision to all the *mixed* cells. However, with each level of the subdivision, the number of cells increases quickly. Considering the fact that not all 'uncertain' regions contribute to separating $\mathbf{q}_{init}$ from $\mathbf{q}_{goal}$, we employ the first-cut algorithm [26] to first subdivide some of the 'uncertain' regions. More specifically, all the *mixed* cells on L are assigned higher priority for the next level of subdivision. Our algorithm is recursively applied until it finds a collision-free path or concludes path non-existence.

4 Cell Labelling

In this section, we present our algorithm to classify the cells in $\mathcal{C}$ using our *free cell query* and *C-obstacle cell query* algorithms. Formally speaking, the free cell query checks whether a given cell C is *empty* or the following predicate P_f is true:

$$P_f(\mathcal{A}, \mathcal{B}, C): \ \ \forall \mathbf{q} \in C, \ interior(\mathcal{A}(\mathbf{q})) \cap interior(\mathcal{B}) = \emptyset,$$

where $\mathcal{A}$ is a robot, $\mathcal{B}$ represents obstacles and the *interior* is the interior of a set. Similarly, the C-obstacle cell query checks whether a given cell C is *full* or the following predicate P_o is true:

$$P_o(\mathcal{A}, \mathcal{B}, C): \ \ \forall \mathbf{q} \in C, \ interior(\mathcal{A}(\mathbf{q})) \cap interior(\mathcal{B}) \neq \emptyset.$$

The collision detection algorithms can check whether a specific configuration lies in $\mathcal{F}$ or $\mathcal{O}$. However, we need to check whether a spatial cell lies in $\mathcal{F}$ or $\mathcal{O}$. This query is relatively non-trivial as compared to checking a specific configuration.

In order to perform cell queries, we place the robot at $\mathbf{q}_c$ - the center of the cell and compute the 'extent' of the motion that the robot can undergo as it moves away from $\mathbf{q}_c$ while being confined within the cell C. To answer the predicate P_f, we need to compute another metric: the Euclidean distance between the robot $\mathcal{A}(\mathbf{q}_c)$ and the obstacle $\mathcal{B}$. This metric describes the 'clearance' between the robot and the obstacle. If this 'clearance' is greater than the amount of a motion that the robot can make, the robot will not collide with the obstacle, and the cell C will be declared as a free cell.

In order to perform the C-obstacle cell query, we measure the amount of inter-penetration between the robot and the obstacle, and compare it with the extent of the robot's bounding motion. In the following subsection, we present a definition and a formulation for the inter-penetration between the robot and the obstacle.

4.1 Motion Bound Calculation

In this section, we present bounds on the motion of the robot along a line segment and a cell in C-space.

Bounding motion for a line segment

In order to formulate the bounding motion for a C-space cell, we first deal with a case when a robot moves along a line segment in C-space. Schwarzer *et al.* [20] define the bounding motion λ when a robot moves along a line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$ as:

$$\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = \text{Upper Bound}(\mu(\mathbf{p}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) \mid \mathbf{p} \in \mathcal{A}).$$

For 2D planar robots with translational and rotational DOFs, the bounding motion λ can be computed as a weighted sum of the difference between $\mathbf{q}_a$ and $\mathbf{q}_b$ for translational components x, y and the rotational component ϕ:

$$\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = |\mathbf{q}_{b.x} - \mathbf{q}_{a.x}| + |\mathbf{q}_{b.y} - \mathbf{q}_{a.y}| + R_\phi \times |\mathbf{q}_{b.\phi} - \mathbf{q}_{a.\phi}|,$$

where the weight R_ϕ is defined as the maximum Euclidean distance between every point $\mathbf{p}$ on $\mathcal{A}$ and the rotation center.

It is not difficult to prove we can achieve a tighter bound:

$$\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) = \sqrt{|\mathbf{q}_{b.x} - \mathbf{q}_{a.x}|^2 + |\mathbf{q}_{b.y} - \mathbf{q}_{a.y}|^2} + R_\phi \times |\mathbf{q}_{b.\phi} - \mathbf{q}_{a.\phi}|. \quad (1)$$

For 3D rigid objects, this equation can be easily extended.

Bounding Motion for a Cell

Now, we define the bounding motion λ of a robot when it is restricted within a cell C instead of a line segment:

$$\lambda(\mathcal{A}, C) = \max\{\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b}) \mid \mathbf{q}_b \in \partial C\}, \quad (2)$$

where $\mathbf{q}_a$ is the center of C, and $\mathbf{q}_b$ is any point on ∂C, the boundary of C.

Among all line segments $\pi_{\mathbf{q}_a,\mathbf{q}_b}$, the diagonal line segments have the maximum difference on each configuration component. According to Eq. (1), the maximum of the bounding motion $\lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_b})$ is achieved by any diagonal line segment of the cell. Therefore, the bounding motion for the cell C is equivalent to the bounding motion over any diagonal line segment $\pi_{\mathbf{q}_a,\mathbf{q}_c}$:

$$\lambda(\mathcal{A}, C) = \lambda(\mathcal{A}, \pi_{\mathbf{q}_a,\mathbf{q}_c}), \quad (3)$$

where $\mathbf{q}_a$ is the center of the cell and $\mathbf{q}_c$ is any corner vertex of the cell.

4.2 C-obstacle Cell Query

In order to perform the C-obstacle cell query, we can measure the amount of inter-penetration between the robot and the obstacle, and compare it with the amount of the robot's bounding motion. If the robot only has translation DOFs, we can use translational PD, PD^t, which is defined as the minimum translational distance to separate the robot from the obstacle:

$$PD^t(\mathcal{A}, \mathcal{B}) = \min(\{\| \mathbf{d} \| \mid interior(\mathcal{A} + \mathbf{d}) \cap \mathcal{B} = \emptyset\}).$$

However, PD^t considers only the translation motion to separate the robot from the obstacle. In fact, we can easily show an example that the robot can become disjoint from the obstacle with 'lesser' amount of motion, when we also take into account the rotational motion of the robot. Fig. 3 illustrates such an example. Therefore, PD^t is applicable for C-obstacle cell query, when the robot has only translational DOFs.

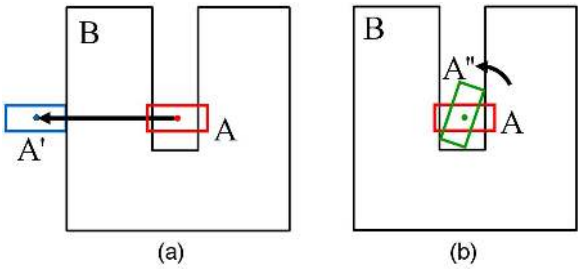

Fig. 3. *An example shows that, to separate A from B, the amount of the 'motion' when both translation and rotation transformation are allowed (a) is much smaller than the amount of the 'motion' when only translation is allowed (b).*

Generalized Penetration Depth

In order to deal with a robot with translational and rotational DOFs, we adopt the notion of generalized penetration depth PD^g proposed by [24], which takes both translational and rotational motion into account. The generalized PD can be defined using the notion of the separating path. A separating path l in C-space is defined as such a curve when a robot moves along l, the robot can be completely separated from the obstacle.

Given a set L of all possible candidates of separating paths, PD^g between a robot $\mathcal{A}$ and an obstacle $\mathcal{B}$ is defined as:

$$PD^g(\mathcal{A},\mathcal{B}) = \min(\{\max(\{\mu(\mathbf{p},l)|\mathbf{p}\in\mathcal{A}\})|l\in L\}). \tag{4}$$

A useful property related to PD^g is as follows:

Lemma 1. *For two convex polytopes $\mathcal{A}$ and $\mathcal{B}$, we have*

$$PD^g(\mathcal{A},\mathcal{B}) = PD^t(\mathcal{A},\mathcal{B}).$$

The proof of this lemma can be found in [24]. Furthermore, like PD^t, PD^g satisfies the property: $PD^g(\mathcal{A},\mathcal{B}) = 0$ if and only if $\mathcal{A}$ and $\mathcal{B}$ are disjoint.

The exact computation of PD^g between non-convex objects is difficult [24]. In our C-obstacle query algorithm, we compute a lower bound on PD^g, which guarantees the correctness of the query. Using Lemma 1, we efficiently compute a lower bound on PD^g by (1) decomposing non-convex models into convex pieces and (2) for each convex pair, compute the PD^t as its PD^g, (3)take the maximum value of PD^gs between all pairwise combinations of convex pieces. Many algorithms are known to compute the PD^t between two convex polytopes [6, 22, 13]. The resulting PD^g algorithm is described in Algorithm 2.

4.3 C-obstacle Query Criterion

We now state a sufficient condition for C-obstacle cell query; i.e., checking whether $\mathcal{A}$ and $\mathcal{B}$ overlap at every configuration $\mathbf{q}$ within a cell C.

Theorem 1: *For a cell C with a center at $\mathbf{q}_a$, the predicate $P_o(\mathcal{A},\mathcal{B},C)$ is true if:*

Algorithm 2 Lower bound of PD^g computation
Input: The robot $\mathcal{A}$, the obstacle $\mathcal{B}$ and the configuration $\mathbf{q}$
Output: The lower bound of PD^g between $\mathcal{A}(\mathbf{q})$ and $\mathcal{B}$.

```
{During preprocessing}
Decompose A and B into m and n convex pieces; i.e., A = ∪A_i and B = ∪B_j.
{During run-time query}
for each pair of (A_i(q), B_j) do
    k = (i − 1)n + j
    if A_i(q) collides with B_j then
        PD^g_k = PD^t((A_i(q), B_j)
    else
        PD^g_k = 0
    end if
end for
return max(PD^g_k) for all k.
```

$$PD^g(\mathcal{A}(\mathbf{q}_a), \mathcal{B}) > \lambda(\mathcal{A}, C). \tag{5}$$

Proof. Our goal is to show that Eq. (5) implies that there is no free configuration along any line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$, where $\mathbf{q}_b$ is any configuration on the boundary of the cell C. According to the definition of PD^g, the maximum trajectory length for every point on a robot $\mathcal{A}$ moving along a possible separating path should be greater than or equal to $PD^g(\mathcal{A}(\mathbf{q}_a), \mathcal{B})$. Moreover, according to Eq. (2), the trajectory length of the robot when it moves along $\pi_{\mathbf{q}_a,\mathbf{q}_b}$ is less than or equal to $\lambda(\mathcal{A}, C)$. Since $PD^g(\mathcal{A}(\mathbf{q}_a), \mathcal{B}) > \lambda(\mathcal{A}, C)$, the minimum motion required to separate the robot $\mathcal{A}$ from obstacle $\mathcal{B}$ is larger than the maximum motion the robot $\mathcal{A}$ could make. Therefore, there is no free configuration along any line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$.

Since there is no free configuration along every line segment between $\mathbf{q}_a$ to $\mathbf{q}_b$, this concludes that every configuration in the cell C lies inside C-obstacle space, and therefore, the predicate $P_o(\mathcal{A}, \mathcal{B}, C)$ holds. □

We use *Theorem* 1 to conservatively decide whether a given cell C lies inside C-obstacle space. The C-obstacle query algorithm includes two parts:

1. Compute a lower bound on PD^g for the robot $\mathcal{A}(\mathbf{q}_a)$ and the obstacle $\mathcal{B}$, which is computed by Algorithm 2.
2. Compute an upper bound on motion, $\lambda(\mathcal{A}, C)$, which can be easily computed by Eq.'s (3) and (1).

Our C-obstacle cell query algorithm is general for both 2D and 3D rigid objects. We have implemented the query for both types of objects. The main computational component is to compute PD^t between convex objects, which can be performed efficiently.

4.4 C-free Query Criterion

Similarly to C-obstacle cell query, in order to check whether a cell is free, we compare the Euclidean distance between the robot $\mathcal{A}(\mathbf{q}_c)$ and the obstacle $\mathcal{B}$ with the bounding motion of the cell $\lambda(\mathcal{A}, C)$. If the distance is greater than the bounding motion, then the cell is classified as a free cell.

4.5 Extension to Articulated Robots

Our method to check for path non-existence, based on C-free and C-obstacle queries, can be extended for articulated robots. The main issues for articulated robots are modifications in the following algorithms: generalized penetration depth, Euclidean distance, and bounding motion computations.

The definition of generalized penetration depth PD^g in Eq. (4) is also applicable to articulated robots. In this case, the separating path in C-space is defined as a curve such that when the articulated robot $\mathcal{A}$ moves along it, $\mathcal{A}$ will be completely separated from the obstacle. In order to compute a lower bound of PD^g between $\mathcal{A}$ and the obstacles, we regard each link of $\mathcal{A}$ as a rigid robot with translational and rotational DOFs. The maximum of lower bounds PD^g between each link of $\mathcal{A}$ and the obstacles yields a lower bound on PD^g between $\mathcal{A}$ and the obstacles. In order to compute the Euclidean distance and bounding motion for the articulated robots, we use the algorithms introduced by [20].

5 Experimental results

In this section, we describe the implementation of our algorithm and highlight its performance on several motion planning scenarios. All timings are measured on a 2.8 GHz Pentium IV PC with 2G RAM. Our implementation is not optimized.

We illustrate the running process of our algorithm for the 'two-gear' example in Fig. 4. In order to find whether the gear-shaped robot can pass through the passage among star-shaped obstacles, the algorithm builds the connectivity graph G for *empty* and *mixed* cells as well as a subgraph of G for *empty* cells. The cell decomposition, induced by guiding path from the connectivity graph, is iterated by 40 times until the initial and goal configurations are found to be separated by *full* cells. The entire process in our algorithm takes 3.356s.

We have applied our algorithm on more complex examples of: 'five-gear', 'five-gear with narrow passage', '2D puzzle' and '2D puzzle with narrow passage'. Tab. 1 highlights the performance of our algorithm on these examples. According to Tab. 1, our approach can report the path non-existence of these examples within 10s. In particular, for the 'five-gear' example, the total timing is 6.317s, while the C-obstacle and C-free queries takes about 1.162s and 1.376s, respectively.

Tab. 2 lists various statistic information of our algorithm on different examples. For the 'five-gear' example, the c-space subdivisions based on the

	two-gear	five-gear	five-gear,narrow	puzzle	narrow puzzle
Total timing(s)	3.356	6.317	85.163	1.555	16.051
C-free Query(s)	0.858	1.376	6.532	0.675	4.556
C-obstacle Query(s)	0.827	1.162	4.675	0.466	2.609
G searching(s)	0.389	1.409	30.687	0.140	3.184
G_e searching(s)	0.077	0.332	7.169	0.044	0.824
Subdivision,Overhead(s)	1.205	2.038	36.100	0.530	4.878

Table 1. Timing of our method: Our approach can report the path non-existence of the examples in less than 10s.

guiding path are iterated for 67 time. The final cell-decomposition includes 168008 cells, including 15324 *empty* cells, 74713 *full* cells and 77971 *mixed* cells.

	two-gear	five-gear	narrow five-gear	puzzle	narrow puzzle
# of iterations	41	67	237	30	93
# of C-free queries	32329	44649	192009	14038	92409
# of C-obstacle queries	30069	41177	176685	16297	85014
# of cells	28288	39068	168008	14260	80858
# of free cells	2260	3472	15324	2259	7395
# of c-obstacle cells	12255	16172	74713	4633	34046
# of mixed cells	13773	19424	77971	7368	39417

Table 2. Statistical information of our algorithm.

Since our algorithm is built on cell decomposition, the algorithm is applicable to finding a collision-free paths even when a narrow passage exists. Finding a collision-free path through a narrow passage has been considered as a difficult task for probabilistic methods, such as PRM. Fig. 7 shows such an example, and Tab. 1 highlights the performance of our algorithm. Also, for this example, our method can achieve about 1.3 time speedup over a deterministic sampling approach, the star-shaped roadmap [25].

5.1 Comparison

We compare our algorithm for path non-existence with star-shaped roadmap algorithm, especially because our approach shares similarities with the star-shaped roadmap algorithm. Their method partitions the free space into star-shaped regions such that a single point called the guard can see every point in the star-shaped region. In our approach, the *empty* cells are a special case of star-shaped regions where the centroid of our cell can be always considered as a guard. Moreover, our method can label *empty* cells much more simply than the star-shaped roadmap, as the star-shaped roadmap is based on expensive, contact surfaces enumeration.

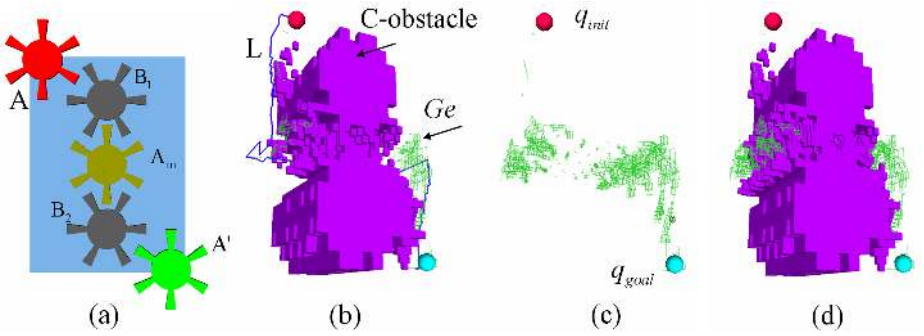

Fig. 4. *The snapshots of the process of our algorithm. (a) The goal of this example is to move a gear-shaped robot from A to A′ through the two gear-shaped obstacles B_1 and B_2. It is uncertain whether there is a path for this problem, even though the robot at A_m is collision-free. (b) shows the graph G_e built from empty cells, and the region of full cells (shaded volumes). Since there exists no path in this graph, we use the guiding path L to indicate the next level of subdivision. (c, d) After the subdivision is recursively applied, the algorithm finally finds that indeed no path exists. This is because the initial and goal configurations are separated by full cells (shaded volumes). (d) also highlights that, to find path non-existence for this example, it is unnecessary to subdivide the entire configuration space.*

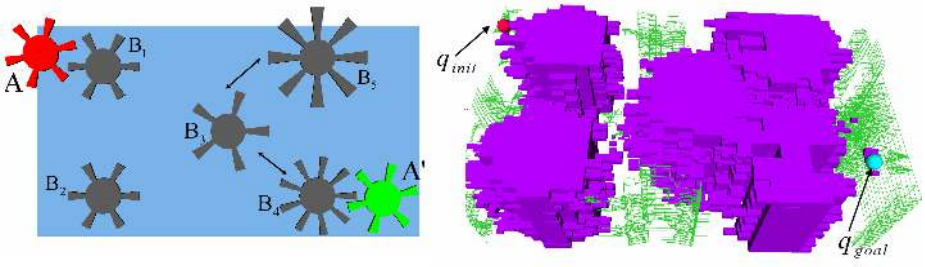

Fig. 5. *'Five-gear' example. (Left) The goal of this example is to move a gear-shaped robot from A to A′ through the five gears B_1, ... and B_5. (Right) There does not exist a collision-free path for this example. This is because the initial and goal configurations are separated by full cells, which correspond to shaded volumes. The right figure also highlights that to find path non-existence for this example, it is unnecessary to subdivide the entire configuration space.*

Finally, their method needs to explicitly capture the intra-connectivity between two adjacent regions, which can be computed using a similar way to the guard computation, but in one dimension less. In our method, the intra-connectivity between two adjacent *empty* cells is implicit and an edge connecting the centroids of two adjacent cells can represent such a connectivity. All these simplicities make our approach extend for a high-DOF problem.

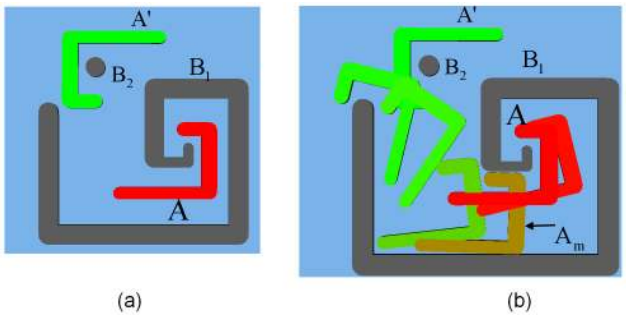

Fig. 6. *'2D puzzle' example. (a) Our algorithm can report the path non-existence for the problem of A reaching A′ in* 1.855*s. (b) is a modified version of (a) with the obstacle* B_1 *enlarged. Our algorithm can find a collision-free path through a narrow passage among the obstacles. The intermediate configurations of the robot along the collision free path, such as* A_m*, are also displayed.*

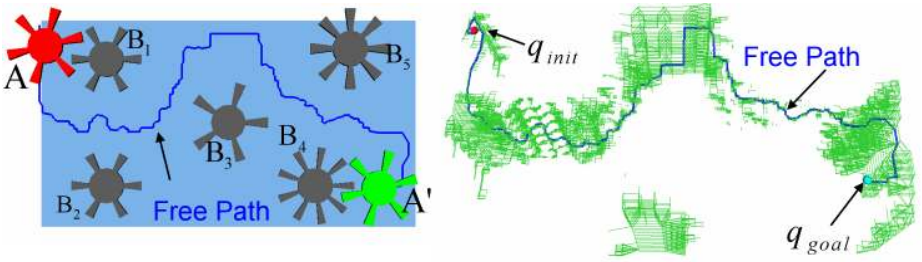

Fig. 7. *Finding a narrow passage for the modified 'five-gear' example from Fig. 5. (Left) this planning problem is almost the same as Fig. 5 except that the obstacle* B_5 *is slightly modified as well as translated. (Right) our method can find a path under the existence of narrow passages, which are challenging for probabilistic methods, such as PRM. The collision-free path, passing through the narrow passage in free space, is derived from empty cells.*

5.2 Analysis

The computational complexity of our C-obstacle query is *asymptotically tightly bound* by the general penetration depth PD^g computation. The computational complexity of the lower bound PD^g used by our algorithm is determined by the number of convex pieces decomposed from the robot $\mathcal{A}$ and the obstacle $\mathcal{B}$, and the geometric complexity of these convex pieces, which is determined by the total number of features of the resulting convex pieces. Let m, n as the number of the convex pieces of $\mathcal{A}$ and $\mathcal{B}$, respectively. Let the geometric complexity of the convex pieces of $\mathcal{A}$ and $\mathcal{B}$ be a and b, respectively. Then, the average numbers of features in each piece of $\mathcal{A}$ and $\mathcal{B}$ are $\frac{a}{m}$ and $\frac{b}{n}$, respectively. Using computational complexity of translational PD, we can derive that the computational complexity of PD^g for 2D rigid objects $\mathcal{A}$ and $\mathcal{B}$ is $O(an + bm)$, and for 3D rigid objects is $O(ab)$.

Our algorithm for checking path non-existence performs an adaptive subdivision of the configuration space. At each step, the number of subdivisions depends on the number of the *mixed* cells indicated by the guiding path L.

5.3 Limitations

Our approach has a few limitations. Our C-free and C-obstacle queries are conservative, which stems from the conservativeness of PD^g and bounding motion computations. Secondly, our algorithm assumes that are no tangential contacts in the boundary of the free space, otherwise, our path non-existence algorithm may not terminate. As a result, our algorithm can not deal with compliant motion planning, where a robot cannot pass through obstacles when the robot is not allowed to touch them. The complexity of our adaptive subdivision algorithm varies as a function of the dimension of the configuration space. Our current implementation is limited to 3-DOF robots.

6 Conclusion and Future work

In this paper, we present an simple approach to check for path non-existence for low DOF robots. Our approach uses two basic queries to efficiently check whether a cell in c-space lies entirely inside free space (C-free cell query) or inside C-obstacle space (C-obstacle cell query). We describe simple and efficient algorithms to perform C-free and C-Obstacle queries using separation and *generalized penetration distance* computations. Our approach is much easier and simpler than prior methods based on cell decomposition. This simplicity enables us to extend our approach to high-DOF motion planning problems.

There are several directions to pursue for future work. We will like to extend our approach for a higher DOF robots. We are also interested in combining our algorithm with probabilistic sampling algorithms design a hybrid planner, which is not only able to find a collision-free path, but can also handle narrow passages and check for path non-existence.

References

1. Marco A. Morales A., Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M. Amato. C-space subdivision and integration in feature-sensitive motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, 2005.
2. N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. Obprm: An obstacle-based prm for 3d workspaces. *Proceedings of WAFR98*, pages 197–204, 1998.
3. Francis Avnaim and J.-D. Boissonnat. Practical exact motion planning of a class of robots with three degrees of freedom. In *Proc. of Canadian Conference on Computational Geometry*, page 19, 1989.
4. J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen. Disconnection proofs for motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, 2001.

5. R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Trans. Syst*, SMC-15:224–233, 1985.
6. S. Cameron. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *IEEE International Conference on Robotics and Automation*, pages 3112–3117, 1997.
7. J.F. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press, 1988.
8. B. R. Donald. Motion planning with six degrees of freedom. Master's thesis, MIT Artificial Intelligence Lab., 1984. AI-TR-791.
9. D. Halperin. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232, 2002.
10. D. Hsu, L. Kavraki, J. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. *Proc. of 3rd Workshop on Algorithmic Foundations of Robotics*, pages 25–32, 1998.
11. L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, pages 12(4):566–580, 1996.
12. K. Kedem and M. Sharir. An automatic motion planning system for a convex polygonal mobile robot in 2-d polygonal space. In *ACM Symposium on Computational Geometry*, pages 329–340, 1988.
13. Y.J. Kim, M. Lin, and D. Manocha. Deep: Dual-space expansion for estimating penetration depth between convex polytopes. In *Proc. IEEE International Conference on Robotics and Automation*, May 2002.
14. J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
15. T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.
16. T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.
17. C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In *Proc. of 4th International Workshop on Algorithmic Foundations of Robotics*, 2000.
18. E. Plaku, K. E. Bekris, B.Y. Chen, A. M. Ladd, and L. E. Kavaraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 2005.
19. J. T. Schwartz and M. Sharir. On the piano movers probelem ii, general techniques for computing topological properties of real algebraic manifolds. *Advances of Applied Maths*, 4:298–351, 1983.
20. F. Schwarzer, M. Saha, and J.C. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Tr. on Robotics*, 21(3):338–353, June 2005.
21. T. Simeon, J. P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000.
22. G. van den Bergen. Proximity queries and penetration depth computation on 3d game objects. *Game Developers Conference*, 2001.
23. Gokul Varadhan and Dinesh Manocha. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.
24. L. Zhang, Y.J. Kim, G. Varadhan, and D. Manocha. Generalized penetration depth computation. Technical Report TR05-026, University of North Carolina at Chapel Hill, 2005.

25. L. Zhang, Y.J. Kim, G. Varadhan, and D. Manocha. Fast c-obstacle query computation for motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, 2006.
26. D. Zhu and J. Latombe. Constraint reformulation in a hierarchical path planner. *Proceedings of International Conference on Robotics and Automation*, pages 1918–1923, 1990.
27. D. Zhu and J. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. on Robotics and Automation*, 7(1):9–20, 1991.