A Simple Path Non-Existence Algorithm using C-obstacle Query

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

Abstract: 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 configuration space or C-space. We use two basic queries: free cell query, which checks whether a cell in C-space lies entirely inside the free space, and C-obstacle cell query, which checks whether a cell lies entirely inside the C-obstacle region. 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 region. We describe simple and efficient algorithms to perform free cell and C-obstacle cell queries using *separation distance* and *generalized penetration depth* 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 has been 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 [7, 14, 15]. In particular, planning algorithms such as the roadmap methods and exact cell decomposition are referred 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 are very difficult to implement. Their practical implementations are usually limited to planar robots, 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 based algorithms subdivide the configuration space into cells and can be made *resolution complete* based on a suitable choice

of parameters. However, the number of subdivision in prior approaches grows quickly with the dimension of the configuration space. In practice, prior cell-decomposition algorithms also suffer from the combinatorial complexity and robustness issues with respect to contact surface enumeration and computation. For a robot with high geometric complexity, generating and enumerating contact surfaces can be complicated and time consuming [8, 16].

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 DOF. 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 may 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 and efficient cell decomposition algorithm for path non-existence from the initial to the goal configuration. Our resulting motion planning algorithm is a complete algorithm for a rigid robot with translational and rotational DOF. Furthermore, our approach can also be extended to articulated robots.

We subdivide the configuration space into *empty*, $full$ and *mixed* cells. Unlike prior cell decomposition algorithms, we use efficient algorithms to label a given cell as $full$ or *empty* to check whether it lies entirely in the C-obstacle region or in the free space. Our algorithm uses two kind of queries: *free cell* query for identifying *empty* cells and *C-obstacle cell* query for identifying $full$ cells. We efficiently perform these queries in the workspace by computing the *separation distance* and *generalized penetration depth*. As a result, our cell query algorithms can be easily implemented for 2D or 3D rigid robots, or articulated robots.

In order to check for path non-existence, the algorithm searches for a sequence of adjacent *empty* or *mixed* cells to connect the cell containing the initial configuration to the cell containing the goal configuration. The non-existence of such a sequence is a sufficient condition for path non-existence between the initial and the goal configuration. We have implemented our algorithm and highlight its performance on 3 DOF robots.

Organization

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

2 Previous Work

Motion planning has been extensively studied for more than three decades. Excellent surveys of this topic are available in [7, 14, 15]. 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 [2, 9, 17, 12], roadmap methods [6], and exact cell decomposition methods [20]. The exact cell decomposition methods require an exact description of the configuration space consisting of the free space and the C-obstacle region. The boundary between the free space and C-obstacle region 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 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 and high DOF robots. As a result, many variants have been proposed to deal with special cases of motion planning problems [14].

2.2 Approximate Cell Decomposition and Sampling-based Approaches

A number of algorithms based on approximate cell decomposition have been proposed [4, 8, 27]. These methods partition the C-space into a collection of cells. They classify the cells into three types: *empty* cells that lie entirely in the free space, *full* cells that are entirely within the C-obstacle region, and *mixed* cells that correspond to the rest. Unlike exact cell decomposition, the cell 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.

One of the main computational issues in approximate cell decomposition methods is cell labelling. In order to label a cell, most prior approaches rely on contact surface computations [27], which could be complicated and prone to degeneracies. Paden et al. [18] describe a method based on workspace distance computation. However, their method could be overly conservative in practice.

The probabilistic roadmap method (PRM) [11] is perhaps the most widely used path planning algorithm 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 [1, 10, 19] or use visibility-based techniques [22]. All these methods are *probabilistically complete*.

Recently, a deterministic sampling approach called star-shaped roadmaps has been proposed [24]. 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 the global connectivity of the free space, the number of regions can increase exponentially as a function of DOF. Moreover, this approach is based on 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. Approximate cell decomposition approaches can also check for path non-existence, but they are also complicated because of contact surface computation. 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 [3]. The authors have proposed a *disconnection prover*, probabilistically showing that the motion planning problem has no solution. However, this approach is restricted to the special problem of finding a path through a planar section. The deterministic sampling approach such as star-shaped roadmaps [24] is a complete approach but it may be overly conservative and can generate a large number of samples.

3 Overview

In this section, we give an overview of our algorithm. Following the basic framework of approximate cell decomposition, we use reliable algorithms to perform the cell labelling. Then we build the connectivity graph for the *empty* and *mixed* cells from the cell decomposition, and check for path non-existence between the initial and the goal configuration by performing a search in the connectivity graph.

3.1 Notation

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

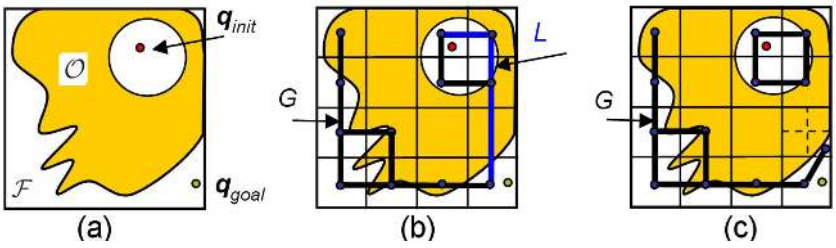

Fig. 1. *Path non-existence between* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$*. (b): A connectivity graph G is built. The path L, which connects the cells including* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$*, is computed from G. Any mixed cell along L is further subdivided. (c): In the new connectivity graph, the cell containing* $\mathbf{q}_{init}$ *and the cell containing* $\mathbf{q}_{goal}$ *are not connected. This concludes that there is no collision-free path between* $\mathbf{q}_{init}$ *and* $\mathbf{q}_{goal}$*.*

$$C = [x'_1, x''_1] \times [x'_2, x''_2] \cdots \times [x'_n, x''_n].$$

We denote $\mathcal{A}(\mathbf{q})$ as a placement of the robot $\mathcal{A}$ at configuration $\mathbf{q}$. Let $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ represent the initial and the goal configuration of the robot. 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 motion curve defined in the C-space. We denote $\mu(\mathbf{p}, l)$ as the trajectory length of a point $\mathbf{p}$ on $\mathcal{A}$ when $\mathcal{A}$ moves along the motion curve 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. 1 illustrates different type of cells. In Section 4, we present more details about our cell labelling algorithm.

3.3 Connectivity Graphs

For each level of subdivision, the connectivity graph G is built to represent the adjacency relationship between *empty* and *mixed* cells. Formally, the connectivity graph [14, 27, 28] 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 sufficient to claim that there is no collision-free path that connects $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$, or $\mathbf{q}_{init}$ and $\mathbf{q}_{goal}$ are not connected.

There are various known techniques to 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} in 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 we can prove path non-existence, or we can find a collision-free path. For this purpose, a subgraph G_e of G is also constructed. G_e represents the adjacency relationship among all the *empty* cells. Intuitively, G_e represents the connectivity of a part of the 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 [27].

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 not connected. If so, we need to further explore the 'uncertain' regions - the union of *mixed* cells, to acquire more information about their connectivity. Considering the fact that not all 'uncertain' regions contribute to separating $\mathbf{q}_{init}$ from $\mathbf{q}_{goal}$, we employ the first-cut algorithm [14, 27] to first subdivide some of the 'uncertain' regions. More specifically, all the *mixed* cells on the path L are assigned higher priorities for the next level of the subdivision than other *mixed* cells. Our algorithm is recursively applied until it finds a collision-free path or concludes path non-existence.

4 Cell Labelling

Compared to prior cell decomposition approaches, one of our distinct features is that during cell decomposition, we use reliable algorithms for cell labelling. As a result, our algorithm does not need to compute the contact surfaces. In this section, we present our *free cell query* and *C-obstacle cell query* algorithms for labelling the cells in $\mathcal{C}$. Formally speaking, the free cell query checks whether a given cell C is *empty* or the following predicate P_f is true:

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

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

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

The collision detection algorithms can check whether a single configuration lies in $\mathcal{F}$ or $\mathcal{O}$. However, these predicates need to check whether a spatial cell lies in $\mathcal{F}$ or $\mathcal{O}$, which corresponds to the collision detection for a set of continuous configurations. Therefore, it is relatively harder to perform these queries as compared to checking a single configuration.

In our algorithms, 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 still being confined within the cell C. To answer the predicate P_f, we compute the *separation distance* between the robot $\mathcal{A}(\mathbf{q}_c)$ and the obstacle $\mathcal{B}$. This distance describes the 'clearance' between the robot and the obstacle. If this 'clearance' is greater than the amount of the maximal 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.

Similarly, 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 subsection 4.2, we shall present our method for formulating and computing the inter-penetration between the robot and the obstacle.

4.1 Motion Bound Calculation

Bounding motion for a line segment

In order to formulate the bounding motion for a C-space cell, we first introduce a case when a robot moves along a line segment in C-space. Schwarzer *et al.* [21] define the bounding motion λ when a robot moves along a line segment $\pi_{\mathbf{q}_a,\mathbf{q}_b}$ as the maximal trajectory length over all points on the moving robot:

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

For 2D planar robots with translational and rotational DOF, 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 on $\mathcal{A}$ and its rotation center. In this case, we can achieve a tighter bound:

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

We can also extend this bound to 3D rigid objects.

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, as:

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

where $\mathbf{q}_a$ is the center of C, and $\mathbf{q}_b$ is any point on ∂C or 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 component between these two configurations. According to Eq. (1), we can infer that 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 $\pi_{\mathbf{q}_a,\mathbf{q}_c}$:

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

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 extent of inter-penetration between the robot and the obstacle, and compare it with a bound on the robot's motion. If the robot only has translational DOF, we can use translational PD, PD^t, which is defined as the minimum translational distance to separate the robot from the obstacle:

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

However, PD^t is only useful to perform C-obstacle cell query, when the robot has only translational DOF. This is because PD^t only considers the translational motion to separate the robot from the obstacle. When the robot is allowed to both translate and rotate, Fig. 2 shows that the robot can be separated from the obstacle with 'less' amount of motion by making use of rotational motion.

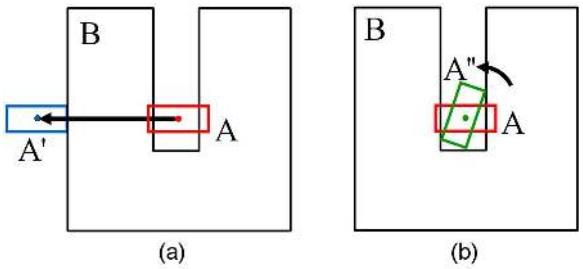

Fig. 2. *An example shows that, to separate A from B, the amount of the 'motion' when both translational and rotational transformation are allowed (b) is much smaller than the amount of the 'motion' when only translation is allowed (a).*

Generalized Penetration Depth

In order to deal with a robot with translational and rotational DOF, we adopt the notion of generalized penetration depth, PD^g, proposed by [25]. PD^g takes both translational and rotational motion into account and can be

defined using the notion of the separating path. A separating path l is such a motion curve in C-space 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:

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

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 [26].

The exact computation of PD^g between non-convex objects is a difficult problem [26]. In our C-obstacle cell 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^g's between all pairwise combinations of convex pieces. Many efficient algorithms are known to compute the PD^t between two convex polytopes [5, 23, 13]. The resulting PD^g computation algorithm is described in Algorithm 1.

Algorithm 1 Lower bound on PD^g computation
Input: The robot $\mathcal{A}$, the obstacle $\mathcal{B}$ and the configuration $\mathbf{q}$
Output: The lower bound on 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.
```

4.3 C-obstacle Cell 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:*

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

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 $\mathcal{A}$ can undergo. Therefore, there are no free configurations 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 $\mathbf{q}_b$, we conclude that every configuration in the cell C lies inside the C-obstacle region, 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 the C-obstacle region. The C-obstacle cell query algorithm consists of two parts:

1. Compute a lower bound on PD^g for the robot $\mathcal{A}(\mathbf{q}_a)$ and the obstacle $\mathcal{B}$ by the Algorithm 1.
2. Compute an upper bound on motion, $\lambda(\mathcal{A}, C)$ 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.

4.4 Free Cell Query Criterion

Similar to C-obstacle cell query, we compare the separation 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 free cell and C-obstacle cell queries based method for checking path non-existence can be extended to articulated robots. The main modifications for articulated robots are in the components: generalized penetration depth, separation 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 on PD^g between $\mathcal{A}$ and the obstacles, we regard each link of $\mathcal{A}$ as a rigid robot with translational and rotational DOF. 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 separation distance and bounding motion for the articulated robots, we use the algorithms introduced by Schwarzer et al. [21].

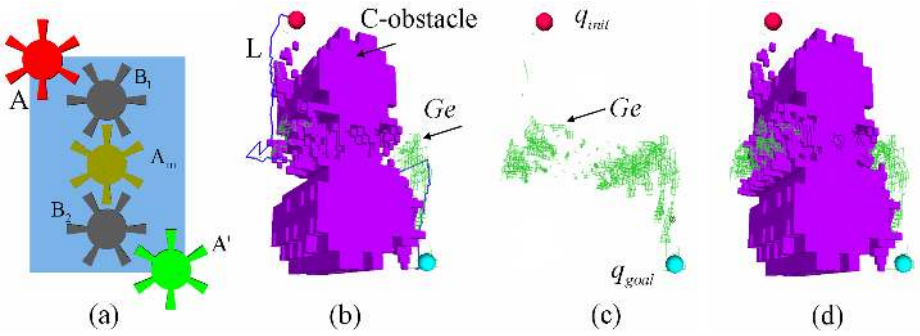

Fig. 3. *Application of our algorithm to the gear benchmark: (a) The goal of this example is to move a gear-shaped robot from A to A' through the two gear-shaped obstacles B_1 and B_2. It is uncertain whether there is a path for these configurations, even though the robot at A_m is collision-free. (b, c) shows the graph G_e built from empty cells, and the region of full cells (shaded volumes). Since no path is found when searching the G_e, we search the graph G for a guiding path L, which indicates the next level of subdivision. (d) After the subdivision is recursively applied, the algorithm finally concludes that no path exists. This is because the initial and the goal configuration are separated by full cells (shaded volumes in (d)).*

	two-gear	five-gear	five-gear,narrow	puzzle	narrow puzzle
Total timing(s)	3.356	6.317	85.163	7.898	15.751
Free cell query(s)	0.858	1.376	6.532	2.174	2.993
C-obstacle cell query(s)	0.827	1.162	4.675	2.021	2.612
G searching(s)	0.389	1.409	30.687	1.991	5.685
G_e searching(s)	0.077	0.332	7.169	0.309	1.035
Subdivision,Overhead(s)	1.205	2.038	36.100	1.403	3.426

Table 1. Performance: This table highlights the performance of our algorithm on different benchmarks.

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 current implementation is not optimized.

We illustrate the running process of our algorithm for the 'two-gear' example in Fig. 3. In order to find whether the gear-shaped robot can pass through the passage among star-shaped obstacles, the algorithm performs cell decomposition, and builds the connectivity graph G for *empty* and *mixed* cells as well as its subgraph G_e for *empty* cells. The cell decomposition, which is performed in the region indicated by the guiding path from the search on the connectivity graph G, is iterated by 40 times until the initial and the goal configuration are found to be separated by *full* cells. The entire computation takes 3.356s.

	two-gear	five-gear	narrow five-gear	puzzle	narrow puzzle
# of iterations	41	67	237	66	107
# of free cell queries	32329	44649	192009	59121	77297
# of C-obstacle cell queries	30069	41177	176685	55683	70438
# of cells	28288	39068	168008	51731	67635
# of empty cells	2260	3472	15324	3438	6859
# of full cells	12255	16172	74713	26295	30351
# of mixed cells	13773	19424	77971	21998	30425

Table 2. Application of our algorithm to different benchmarks

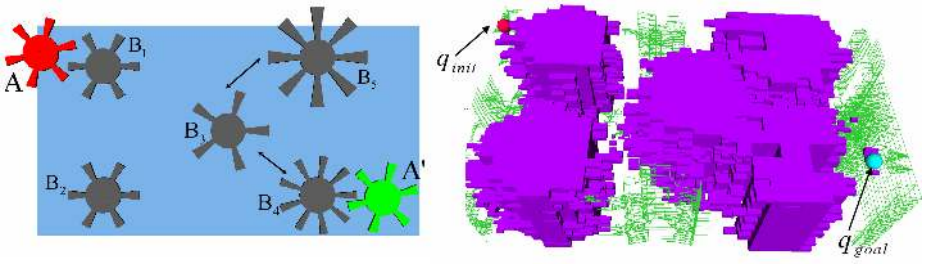

Fig. 4. *'Five-gear' example. (Left) The goal of this example is to move a gear-shaped robot from A to A′ through the five gears B_1, ... and B_5. (Right) There does not exist a collision-free path for this example. This is because the initial and the goal configuration are separated by full cells, which correspond to shaded volumes. The right figure also highlights that to find path non-existence for this example, it is unnecessary to classify the entire configuration space.*

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

Table 2 gives details about application of our algorithm to different benchmarks. For the 'five-gear' example, the cell decomposition, which is restricted in the region indicated by the guiding path, is iterated 67 times. The final cell-decomposition includes 39068 cells, with 3473 *empty* cells, 16172 *full* cells and 19424 *mixed* cells.

Since our algorithm uses 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. 6 shows such an example. According to the Table 1, for this example, our un-optimized method achieves about 1.3 times speedup over a deterministic sampling approach, the star-shaped roadmap [26].

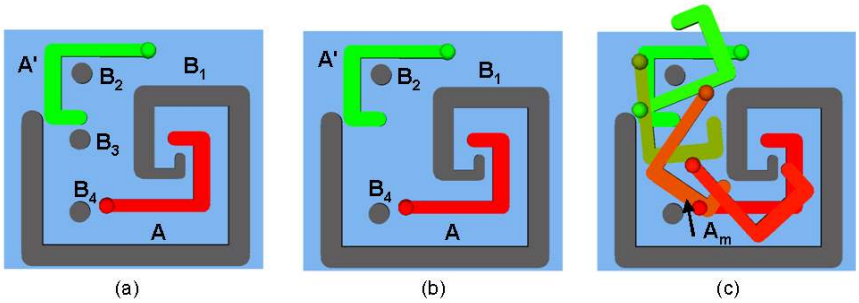

Fig. 5. *'2D puzzle' example. (a) Our algorithm can report the path non-existence for the problem to move A to A′ in 7.898s. (b) is a modified version of (a) without the obstacle B_3. Our algorithm can find a collision-free path through a narrow passage among the obstacles. (c) shows intermediate configurations A_m of the robot along the collision free path.*

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. Star-shaped roadmap method partitions the free space into star-shaped regions and for each star-shaped region computes a single point called a guard which can see every point in the region. In our approach, the *empty* cells are a special case of star-shaped regions where any configuration in the cell can be always considered as a guard. Moreover, our method can label *empty* cells in a simpler way than the star-shaped roadmap, as the star-shaped roadmap is based on expensive contact surfaces enumeration.

Finally, star-shaped roadmap 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 guards of two adjacent cells can represent such a connectivity.

5.2 Analysis

The computational complexity of our C-obstacle cell query is bounded by generalized penetration depth PD^g computation. We only compute a lower bound to PD^g and its complexity is governed by the number of convex pieces that are obtained from the convex decomposition, and the geometric complexity of these convex pieces. Let m, n denote the number of convex pieces of the robot $\mathcal{A}$ and the obstacle $\mathcal{B}$, respectively. Let the geometric complexity of all 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 com-

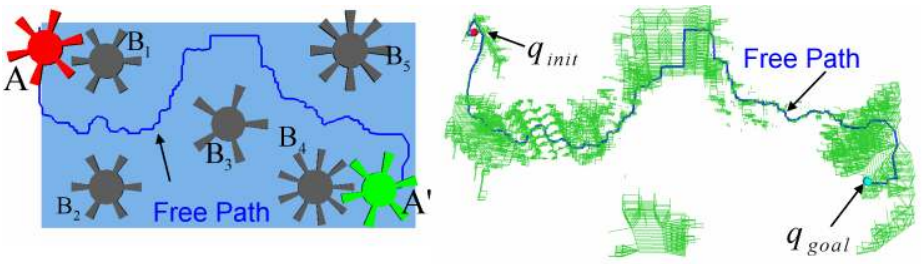

Fig. 6. *Finding a passage through narrow passages for the modified 'five-gear' example. (Left) this planning problem is almost the same as Fig. 4 except that the obstacle B_5 is slightly modified as well as translated. (Right) our method can find a path under the existence of narrow passages, which are challenging for probabilistic methods, such as PRM. The collision-free path, passing through the narrow passage in the free space, is derived from empty cells.*

putational 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 is based on adaptive decomposition of the configuration space. At each step, the number of decomposition depends on the number of the *mixed* cells indicated by the guiding path L.

5.3 Limitations

Our approach has a few limitations. Our free cell and C-obstacle cell 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 a 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 (free cell query) or inside the C-obstacle region (C-obstacle cell query). We describe simple and efficient algorithms to perform these queries using *separation distance* and *generalized penetration depth* computations. Our query algorithms are general for 2D or 3D rigid robots, or articulated robots. Using these queries, our approach for path non-existence computation is simpler and more efficient than prior cell decomposition methods.

There are several directions to pursue for future work. We are interested in extending our approach for higher DOF motion planning problems, such as

6 DOF rigid robots. Our algorithms to compute various queries are directly applicable and the main challenge is to perform spatial cell decomposition in higher dimensions. Moreover, we are interested in combining our algorithm with probabilistic sampling algorithms to design a hybrid planner, which is not only able to find a collision-free path, but can also check for path non-existence and handle narrow passages.

Acknowledgment

This project was supported in part by ARO Contracts DAAD19-02-1-0390 and W911NF-04-1-0088, NSF awards 0400134 and 0118743, ONR Contract N00014-01-1-0496, DARPA/RDECOM Contract N61339-04-C-0043 and Intel. Young J. Kim was supported in part by the grant 2004-205-D00168 of KRF, the STAR program of MOST and the ITRC program.

References

1. 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.
2. F. 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.
3. 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.
4. 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.
5. S. Cameron. Enhancing GJK: Computing minimum and penetration distance between convex polyhedra. *IEEE International Conference on Robotics and Automation*, pages 3112–3117, 1997.
6. J. Canny. *The Complexity of Robot Motion Planning*. ACM Doctoral Dissertation Award. MIT Press, 1988.
7. H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion : Theory, Algorithms, and Implementations*. The MIT Press.
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. 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. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
15. S. M. LaValle. *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006.
16. T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. Comput.*, C-32:108–120, 1983.
17. T. Lozano-Pérez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Comm. ACM*, 22(10):560–570, 1979.
18. B. Paden, A. Mess, and M. Fisher. Path planning using a jacobian-based freespace generation algorithm. In *Proceedings of International Conference on Robotics and Automation*, 1989.
19. 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.
20. 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.
21. F. Schwarzer, M. Saha, and J. Latombe. Adaptive dynamic collision checking for single and multiple articulated robots in complex environments. *IEEE Tr. on Robotics*, 21(3):338–353, June 2005.
22. T. Simeon, J. P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*, 14(6), 2000.
23. G. van den Bergen. Proximity queries and penetration depth computation on 3d game objects. *Game Developers Conference*, 2001.
24. G. Varadhan and D. Manocha. Star-shaped roadmaps - a deterministic sampling approach for complete motion planning. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005.
25. L. Zhang, Y. Kim, G. Varadhan, and D.Manocha. Generalized penetration depth computation. In *ACM Solid and Physical Modeling Symposium (SPM06)*, pages 173–184.
26. L. Zhang, Y. Kim, G. Varadhan, and D.Manocha. Fast c-obstacle query computation for motion planning. In *IEEE International Conference on Robotics and Automation (ICRA 2006)*, pages 3035–3040, 2006.
27. D. Zhu and J. Latombe. Constraint reformulation in a hierarchical path planner. *Proceedings of International Conference on Robotics and Automation*, pages 1918–1923, 1990.
28. D. Zhu and J. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. on Robotics and Automation*, 7(1):9–20, 1991.