

Planning Motion in Point-Represented Contact Spaces Using Approximate Star-Shaped Decomposition

Jyh-Ming Lien and Yanyan Lu

Abstract—Star-shaped decomposition partitions a shape into a set of star-shaped components. A shape is star shaped if and only if there exists at least one point which can see all the points in the shape. Due to this interesting property, decomposing a configuration space into star-shaped components can be beneficial, e.g., for solving motion planning problem. In this paper, we propose a simple method to decompose the contact space, represented by point set data, into *approximate* star-shaped components. We propose two motion planning methods, one deterministic and one probabilistic, both based on this idea.

I. INTRODUCTION

Motion planning is a problem of finding a path to move a robot from a start configuration to a goal configuration without colliding with obstacles. A key concept in studying motion planning problems is the idea of *configuration space*. The configuration space (denoted as C-space or simply \mathcal{C}) of a given motion planning problem is a set of points that represent all possible configurations of the robot [1]. Free configuration space (\mathcal{C}_{free}) is a subset of a \mathcal{C} which represents all collision-free configurations. Essentially, given a motion planning problem, the goal is to find a sequence of configurations in \mathcal{C}_{free} that connects the start configuration to the goal configuration.

Because of the importance of \mathcal{C}_{free} in solving a motion planning problem, the representation of \mathcal{C}_{free} has been one of the fundamental issues in motion planning. Similar to many geometric models, \mathcal{C}_{free} can be represented by its boundary. The boundary of \mathcal{C}_{free} (denoted as $\partial\mathcal{C}_{free}$) is called the contact space, which represents a set of configurations in which the robot is in contact with obstacles. For robots with low degrees of freedom (usually ≤ 3), $\partial\mathcal{C}_{free}$ can be computed exactly [1]. Unfortunately, the computation of an exact $\partial\mathcal{C}_{free}$ has been shown to be intractable for general motion planning problems [2]. Methods have been proposed to approximate $\partial\mathcal{C}_{free}$, e.g., using octrees [3].

Visibility has been an important tool to explore the topology of \mathcal{C}_{free} . For example, we can use visibility to classify shapes. A shape is star shaped if and only if there exists at least one point which can see all the points in the shape. Due to this interesting property, multiple overlapping star-shaped components covering $\partial\mathcal{C}_{free}$ can easily form a network to represent the topology of \mathcal{C}_{free} . Such a network is usually called a roadmap [1] and the motion planning problem can be solved by connecting the query configurations to the roadmap. However, little work (except [4]) has been attempted to use star-shapes to solve motion planning problems. One of the main reasons is the difficulty of extending this concept to higher (> 3) dimensional spaces.

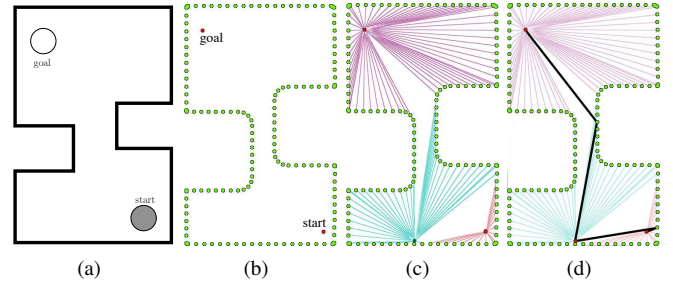


Fig. 1. An overview of our approach. (a) A 2D motion planning problem with a disc robot. (b) Points representing the contact space are generated from the point-based Minkowski sum. (c) An approximate star-shaped decomposition (A^* -DECOMP) with three guards of the points in (b). (d) A path found from the A^* -DECOMP.

To cope with this difficulty, we propose to use *point-represent* $\partial\mathcal{C}_{free}$. The result is a framework that

- 1) represents the $\partial\mathcal{C}_{free}$ using a finite number of points whose resolution (i.e., coverage) is controlled by the user, and
- 2) decomposes these points into a set of approximate star-shaped components (this term will be defined more carefully in Section III).

Fig. 1 shows an overview of our approach. The pseudocode of the proposed framework is sketched in Algorithm I.1.

In this paper, we propose two new methods, one deterministic and one probabilistic. Both methods can be viewed as extensions to the framework described in Algorithm I.1. The deterministic method is resolution complete and is more efficient when applied to problems in low dimensions (≤ 3), while the probabilistic method is more applicable to the problems in high dimensional spaces. The decomposition is done iteratively by generating guards until all points in S are visible. The main differences between the deterministic and the probabilistic methods are how the points cover the $\partial\mathcal{C}_{free}$ and how the visible points are determined. Details of these two methods will be discussed in Sections IV and V.

Algorithm I.1 A^* -ROADMAP

Input. A motion planning problem

Output. A roadmap that captures the connectivity of \mathcal{C}_{free}

Generate a point set S that covers $\partial\mathcal{C}_{free}$

Let $G = \emptyset$ be a set of guards

while exit a point $s \in S$ that is not visible from G **do**

Create a new guard g near s and $G = G \cup g$

Compute the visibility region $V_S(g)$ of g

Extract a roadmap from the visibility of G

Main contributions. To the best of our knowledge, the proposed motion planners are the first planners using the

star-shaped decomposition in both low (≤ 3) and higher dimensional spaces. The first motion planner that decomposes \mathcal{C}_{free} into a set of star-shaped components is due to Varadhan and Manocha [4]. Although we use the same concept of star-shaped decomposition, there are several fundamental and significant differences between their method and our work:

- In [4], meshes are constructed to approximate the contact space while we use only points.
- Because the star-shaped components are constructed on top of an octree in [4], the number of such components can become very large. On the other hand, the number of the star-shaped components is usually small using our method.
- Instead of decomposing \mathcal{C}_{free} , our method decomposes the contact space, which has lower dimensionality and therefore is easier to “guard”.
- Our method can handle general motion planning problems while [4] can only handle problems in 3-d configuration space.

As a result, the proposed method has been shown to naturally provide a (tunable) compact representation of a $\partial\mathcal{C}_{free}$. Our experiments show that the size of the resulting roadmap is usually smaller than the size of the roadmap generated by the sampling-based methods.

II. RELATED WORK

The main feature of our method is the decomposition of the contact space represented by points. In this section, we will review some work closely related to the proposed motion planning method.

Motion Planning. Over the last four decades, many approaches have been proposed to solve this problem; see surveys [1], [5]. Since the motion planning problem has been shown to have high complexity [2], researchers have focused on discrete and approximate approaches. In particular, a collection of randomized methods called *probabilistic roadmap* (PRM) have enabled motion planning to be applied to more realistic applications. PRM methods learn the connectivity of the free space of a given C-space by constructing a graph using random sampling and simple local connections. The solution path is then extracted from the graph.

Star-Shaped Decomposition. There is little known about star-shaped decomposition in space higher than two dimensions. On the contrary, decomposing simple polygons into 2-d star-shaped subpolygons is well studied; see a survey in [6]. Similar to most of the decomposition problems, decomposing a polygon with holes into minimum number of star-shaped components is NP-complete [7]. For polygons without holes, the partitioning can be done in $O(n)$ [8] time and results in $\frac{n}{3}$ components. Star-shaped decomposition is related to guarding an art gallery [9]. A polygon is said to be guarded if it is covered by the visible regions of the guards. The visible region of a guard is a star-shaped component. The problem is known to be NP-complete for polygons with or without holes [10], [11].

Decomposition of Points. Several methods have been proposed to decompose point set data into meaningful components, e.g., by Dey et al. [12]. Recently, Yamazaki et al. [13] proposed a decomposition method based on the estimation of the centrality of each point using approximated geodesic distance. Despite their promising results, one major

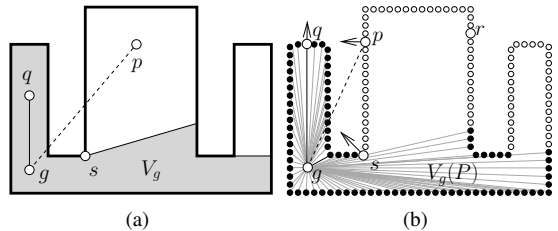


Fig. 2. (a) A polygon. (b) A point set representing the same shape. See Section III for details.

drawback of this approach is that computing the centrality is both time and memory consuming.

In [14], we proposed the only known method that decomposes a set of points into approximate star-shaped decomposition. Our motion planning methods are based on this method, which will be discussed briefly in Section III.

III. PRELIMINARIES

In this section, we will provide a brief overview of the methods that the proposed planner will be based on. To simplify our discussion, we first assume that, in Section III-A, the given point set data represents a general shape and each point in the set is associated with an outward normal. We will discuss how to generate these points in $\partial\mathcal{C}_{free}$ in Section III-B.

A. Preliminary: Point-based Visibility

A point set data is a type of boundary representation except that these points are *not* connected to form meshes. Without connectivity, the boundary of a shape becomes more ambiguous, thus the visibility can be only approximated. Fig. 2 illustrates the differences between the traditional and the point-based visibility.

We let P be a point set. We assume that the point set P is a sample of the boundary of a shape. Let a point $p \in P$ be 2-tuple (v_p, \vec{n}_p) , where v_p and \vec{n}_p are d -dimensional vectors in \mathbb{R}^d , respectively, representing the position and the outward normal direction of the point p .

To compute a star shape, we are mainly interested in if a point g can see points p on the boundary of the shape. More specifically, we find the star shape of g by removing the *invisible points* of g , which include (i) back-facing points of g and (ii) the points that are “blocked” by back-facing points. We observe that if p is a back-facing point of g then the “viewing line” from g to p , denoted as \overline{gp} , and the outward normal at p point are in the opposite direction. The only exception is at the discontinuous point of ∂V_g (e.g., the point s in Fig. 2(b)). The remaining points are the visible points and form a star shape.

Our method is based on this simple observation to find visible points. For a given point g and a point set P , we define $V_g(P)$ as a subset of P that are visible from g .

Finally, we say that the point set P is star-shaped if there exists a guard g such that none of the points p in P are invisible from g . Then, a star-shaped decomposition of P is a set of star-shaped point sets $\{P_i\}$ whose union is P .

Our proposed star-shaped decomposition method is built based on these properties of visibility and internality of point sets without using any global or local connectivity, which can be expensive to compute.

B. Preliminary: Point-based Minkowski Sum

Our method of generating configurations in the contact space is based on the point-based Minkowski sum [15].

The point-based Minkowski sum produces a set of points that *cover* the boundary of the Minkowski sum of two given polyhedra, S and T . More specifically, we will generate a point set P so that P is a d -covering of the Minkowski sum boundary, where d is a user-specified value. Intuitively, d controls the sampling density of a boundary. A smaller d will produce a denser approximation of the boundary.

There are three major steps in the point-based Minkowski sum. First, we sample two point sets from the input S and T . Second, we generate the Minkowski sum of the point sets simply using the definition of the Minkowski sum. Third, we separate the boundary points (both hole and external boundaries) from the internal points.

Step 1: Sample points. Let S and T be two polyhedra. We generate two point sets from S and T , denoted as P_S and P_T . The point set P representing the Minkowski sum boundary of S and T is simply $(P_S \oplus P_T) \cap \partial(S \oplus T)$. We want the point set P to cover the entire Minkowski sum boundary w.r.t. a user specified interval d . As shown in [15], we can guarantee that the final point set is at least a d -covering of the Minkowski sum boundary by simply making P_S and P_T be the d -covering of S and T , respectively.

Step 2: Compute the Minkowski sum. This step is straightforward. Using P_S and P_T , we compute $P_{S \oplus T}$ by simply following the Minkowski sum definition.

Step 3: Extract boundary points. In this final step, we separate (filter) points into two groups: Boundary points and inner points. Boundary points will be returned as our final answer and inner points will be discarded.

The first filter, named *normal filter* determines if a pair of sample points (from S and T , resp.) is an inner point by examining their *origins* and orientations. Kaul and Rossignac [16] have shown that a facet of the Minkowski sum boundary can only come from a facet of S and a vertex from T (or vice versa) or from two edges of S and T , resp., if the facet, vertex and edges are properly oriented [16].

This filter is efficient, but it alone *cannot* filter out all inner points. The second filter, named *CD filter* uses collision detection to separate boundary points from inner points. CD filter is computationally more expensive but it provides an unambiguous decision. More details of these filters can be found in our previous work [15].

IV. DETERMINISTIC A^{ϵ^*} -ROADMAP

In this section, we present a deterministic motion planner, in which both the points representing the contact space and their visibility are computed deterministically. The deterministic method is *resolution complete* and efficiently handles the problems with the dimensionality no higher than three.

A. Generating Points in $\partial\mathcal{C}_{free}$

Recall that in order to apply the concept of point-based visibility sketched in Section III-A, we need both the position and the normal direction of each point in the point set.

The positions of the point set can be obtained using the point-based Minkowski sum discussed briefly in Section III-B. For a robot with rotational degrees of freedom, we compute and enumerate a set of points for each (discrete)

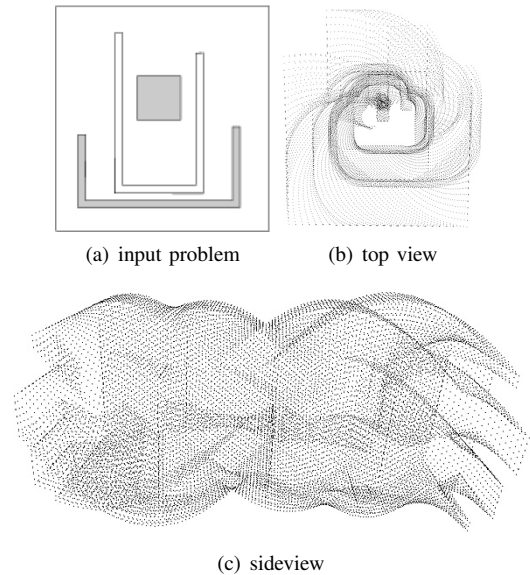


Fig. 3. (a) A 2-d workspace with a robot and the obstacle (shaded). (b)(c) A top and a side views, resp., of the contact space generated from the point-based Minkowski sum. The vertical axis in (c) represents rotational degree of freedom $SO(1)$.

orientation. Because of the simplicity of our framework, these enumerated points do not need to be either connected (e.g., to form a mesh) or ordered. Moreover, since we only consider the problems with low dimensional contact space, the size of the final point size is still reasonable. Figs. 3 shows the points generated from the contact spaces of a motion planning problem.

The orientation (i.e., the outward normal) of the point can also be obtained easily in low dimensional space. Since a point p on the Minkowski sum of S and T can only come from a facet of S or T or from a pair of edges of S and T , respectively, we can quickly determine the normal of p from the normal of the “source” facet or the cross product of the “source” edges.

B. A^{ϵ^*} of a Guard

Based on the properties discussed in the previous section, we are now ready to present the method that approximates the visible region (an *approximate star shape* or A^{ϵ^*}) of a given point. In our planner, we always pick the initial position of a guard g from the points P .

Given a point set P generated in the contact space and a point g , we first cluster P using g . P is clustered in the way that, after projected to the spherical surface around g , nearby points are grouped into a cluster. In our implementation, we project P to the boundary of a $2d$ -sided box, where d is the C-space dimensionality. An example of such a clustering is shown in Fig. 4(a).

Then, for each cluster, we identify the closest invisible point p to g (see Fig. 4(b)). All points in the cluster that are closer than p to g are visible by g . Recall that we can find out if a point is invisible from another by checking the outward normal and the viewing direction. Fig. 4(b) illustrates all the visible points from g , which collectively form the A^{ϵ^*} of g . Algorithm IV.1 outlines this process.

Algorithm IV.1 $A^{\epsilon^*}(P, g)$

Input. A point set, P , and a query point g
Output. A set of points $V_g(P) \subset P$ that are visible from g
Subdivide P into radial subdivisions $\{P_i\}$ centered at g
for each radial subdivision, P_i **do**
 Compute $p \in P_i$, the closest invisible point to g
 if p exists, let $d_p = |g - p|$, **else** $d_p = +\infty$
 for each point $r \in P_i$ **do**
 if $|g - r| < d_p$ **then**
 $V_g(P) \leftarrow r$

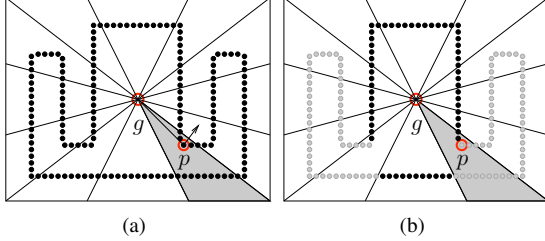


Fig. 4. (a) A radial clustering of the point set P from the point g . Points in P are projected to the boundary of a $2d$ -sided box defined around g . For each cluster, the closest invisible point p is found and all points that are closer than p are considered visible. (b) Darker points are considered as visible points from g .

The advantage of Algorithm IV.1 is its efficiency. Computing a guard takes only $O(n)$ time for a point set with n points and the computation efficiency can be further improved if a spatial data structure is built to pre-process P .

C. A^{ϵ^*} Expansion

In our planner, we pick the guard g from the point set P . For such guards g that are close to the boundary, g can have poor visibility. In this case, more guards are needed to see (cover) the entire point set. Therefore, we need a way to reduce the number of guards needed to cover the space. Intuitively, our strategy is to find a “better g ” near g that can see more points than g .

More specifically, we compute the kernel K_g of the visible points $V_g(P)$ of a guard g and find a new guard in the kernel K_g . Our strategy guarantees to find a larger visibility region than $V_g(P)$. We define a kernel of a point set as the following.

Definition 4.1: A kernel of a set of points Q is another set of points K that is visible from all the points in Q .

Algorithm IV.2 A^{ϵ^*} -EXPAND($g, V_g(P)$)

Compute the kernel K_g from $V_g(P)$
Let K'_g contain $\log n$ vertices from K_g
Find $k \in K'_g$ so that k has the largest visible region $V_k(P)$
if $|V_k(P)| > |V_g(P)| + \frac{|P|}{c}$ **then**
 return A^{ϵ^*} -EXPAND($k, V_k(P)$)
else
 return ($g, V_g(P)$) ▷ no expansion found

Once the kernel is known, we compute a visible region for each vertex in the kernel. Since the visible regions of the vertices of the kernel are larger than or equal to $V_g(P)$, our visible region must expand monotonically. This process is

repeated until no expansion can be gained. The subroutine A^{ϵ^*} -EXPAND is defined in Algorithm IV.2.

The kernel K_g of a set of points V_g can be computed as the intersection of half-spaces defined by the points in V_g . The key to compute the kernel efficiently is that we can convert these half-spaces to the points \overline{V}_g in the dual space, where \overline{V}_g is defined as: $\overline{V}_g = \{\frac{\vec{n}_g}{p \cdot \vec{g} \cdot \vec{n}_p} \mid p \in V_g\}$.

Then the boundary points in the kernel K_g are simply dual to the facets of the convex hull of \overline{V}_g , which can be computed efficiently in $O(n \log n)$, where n is the number of points in V_g . The time complexity of Algorithm IV.2 is $O(n \log n)$ for problems in a low dimensional space.

D. Discussion

To conclude this section, we provide some discussions and implementation issues of the proposed method.

Resolution complete. The motion planner proposed in this section can only be applied to the problems in low dimensions. The main reasons include (i) the number of points required to cover the contact space and (ii) the radial clustering for computing the visibility. Both of these methods cannot be extended to more general motion planning problems without losing efficiency. However, an important benefit of this deterministic planner is its resolution completeness. This is because when the value of the covering d approaches to zero the contact space will be completely covered.

Special orthogonal group. One important detail that requires special attention is that some dimensions of the C-space represent the orientations of the robot. For example, the best way to represent the contact space in Fig. 3(c) is in fact a donut shape that joins the top and the bottom points together. Therefore, during the computation of the A^{ϵ^*} , we should be aware that a configuration in the contact space can have multiple representations (e.g., by adding 2π to the orientation) and can be classified into multiple clusters.

Query. Since the visibility of points is not exact, the connection between configurations may not be always valid. From our experiments, there are roughly 5% of the edges in the roadmap colliding with obstacles. As a result, during the query process, we have to validate the extracted path. When an edge in the path is found invalid, we will simply delete that edge and extract a new path.

Moreover, we can solve the motion planning problem in the fashion of multiple queries or single query. If multiple queries are expected, we simply decompose the points so that all points or a large portion of the points in the contact space are visible by at least one guard. Similar to the star-shaped roadmaps [4], we form a network of the guards to capture the connectivity of the \mathcal{C}_{free} .

If the motion planning problem is a single shot, we do not need to decompose the entire contact space. In this case, we start to grow two A^{ϵ^*} trees rooted at the start and the goal configurations. To expand an existing A^{ϵ^*} tree, we simply compute the A^{ϵ^*} for the farthest visible point by the tree, and include the newly visible points as part of the tree. The process repeats until the goal is visible from one of the guards in the tree.

V. PROBABILISTIC A^{ϵ^*} -ROADMAP

In this section, we propose a probabilistic motion planner that can handle problems in higher dimensional space.

A. Generating Points in $\partial\mathcal{C}_{free}$

Because we only enumerate points from the contact space, our approach can naturally be applied to the motion planning problems involved with articulated robots and multiple free-flying robots using the strategy discussed above. However, enumerating points to form a d -covering becomes unrealistic for these robots. In fact, The number of the points needed to cover the contact space grows exponentially with respect to the degrees of freedom of the robot.

To cope with this problem, we take the strategy similar to PRMs, where the user will be asked to specify the number of points in the final roadmap in \mathcal{C}_{free} . We ask users to specify the number of points needed to represent the contact space in $\partial\mathcal{C}_{free}$. To generate these points, we still use the point-based Minkowski sum and enumerate over the rotational degrees of freedom. However, we randomly throw away points to maintain the size and the variety of the configurations.

Approximate Normals. Recall again that computing visibility requires the point normals to check if the points are invisible from a guard g . Point normals can be estimated exactly when the dimensionality of the problem is low. In high dimensions, we can only approximate the normal direction. Let p be a point in the point set generated as the method discussed above. Because p is in the contact space, a small perturbation in the direction of \overrightarrow{gp} or \overrightarrow{pg} will put the robot in collision with the obstacle, where g is a guard. Thus, we can approximate the normal of p using either \overrightarrow{gp} or \overrightarrow{pg} , whichever pushes p into the \mathcal{C}_{free} .

B. A^{ϵ^*} of a Guard

Recall that, in low dimensions, we compute the point-based visibility by projecting all points onto the surface of a unit sphere centered at the guard g , and then we cluster the points by superimposing a grid in spherical coordinates. In each of these clusters, we find a list of points that are visible from g . Although the definition of the point-based visibility remains the same in high dimensional space, the size of the grid grows exponentially with the degrees of freedom of the robot. As a result, this approach becomes unrealistic.

To cope with this problem, we again allow users to provide the number of clusters as an input parameter k . Then, the planner uses this parameter with a k -means clustering method and Lloyd’s algorithm to cluster the points P . More specifically, we randomly select k configurations $p \in P$ from the point set and use the vectors \overrightarrow{gp} pointing from the guard g to p as the “centers” the clusters. Then each point $q \in P$ is assigned to a cluster whose angles between \overrightarrow{gq} and the center of the cluster is smaller than those of the other clusters. Once we have the clusters, we find a list of points that are visible from g in each of these cluster based on the same definition for the point-based visibility.

VI. DISCUSSION AND COMPARISON

A^{ϵ^*} -based motion planning is closely related to PRM methods. Both of these planning strategies avoid computing the exact representation of the \mathcal{C}_{free} . PRMs have been shown to solve many practical motion planning problems. Nevertheless, PRM-based motion planners are probabilistic and one of the most famous problems called the “narrow passage problem” prohibits PRM from solving some simple problems, e.g., the motion planning problem shown in Fig. 5(a). On

TABLE I
 A^{ϵ^*} -ROADMAP VS. PRMS VS. VISIBILITY PRMS VS.
STAR-SHAPED ROADMAPS

	approximation type	roadmap size	handle > 3 \mathcal{C} -space
A^{ϵ^*} -based roadmap	both	small	easy
PRM [17]	probabilistic	large	very easy
visibility PRM [18]	probabilistic	small	very easy
star-shaped map [4]	deterministic	large	hard

the contrary, the A^{ϵ^*} -based motion planner is resolution complete thus does not suffer from the narrow passages. Another benefit of the A^{ϵ^*} -based motion planner is that it provides a natural way to provide a (tunable) compact representation of the \mathcal{C}_{free} by monotonically expanding the A^{ϵ^*} of a guard.

A^{ϵ^*} -based motion planning is also closely related to the star-shaped roadmaps [4]. Both methods are deterministic and attempt to approximate the contact space using star-shaped decomposition and the Minkowski sum of the robot and the obstacles. However, the representation of the contact space and the way that the Minkowski sum and the star-shaped decomposition are generated are all fundamentally different. In the star-shaped roadmaps, meshes are used to represent the contact space and star-shaped components are identified within the octree cells. On the contrary, the A^{ϵ^*} -based motion planner represents the contact space using point set data and the star-shaped components can be arbitrarily large. Therefore, as a consequence, the A^{ϵ^*} -based motion planner produces fewer guards to cover the contact space. Another advantage of the A^{ϵ^*} -based motion planner over the star-shaped roadmaps is that point-based representation allows us to extend the basic framework to become probabilistic and can handle higher dimensional problems, e.g., the problem in Fig. 5(c). Table I summarizes our comparison to PRMs and the star-shaped roadmaps.

VII. EXPERIMENTAL RESULTS

In this section, we show experimental results of our A^{ϵ^*} -based motion planners. We test our implementation on three motion planning problems, including a 3-d translational robot and a 2-d (shown in Fig. 5). Both of these experiments are conducted using the deterministic version of the planner. We also studied a 3-d free-flying robot using the probabilistic version of the planner. All the experiments reported in this paper are performed on a Pentium 2.0 GHz CPU with 512 MB RAM. Our implementation is done in C++. Results of our experiments are shown in Tables II.

3-d translational robot. The motion planning problem shown in Fig. 5(a) has a robot tightly fit into the hole of the obstacle. Our goal is to remove the robot from the holes. To generate the points covering the contact space, we sample 48 points from the robot and 74 points from the obstacle.

2-d free-flying robot. The motion planning problem shown in Fig. 5(b) has two interlocked ‘P’s. Our goal is to unlock them. To generate the points covering the contact space, we sample 52 points from the robot and 52 points from the obstacle and we compute the Minkowski sum from 100 orientations of the robot.

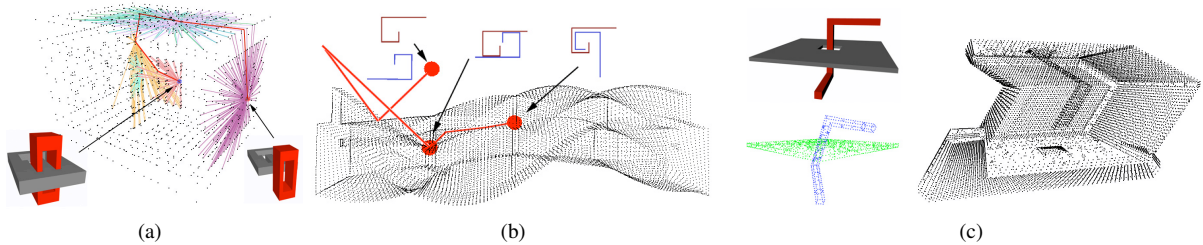


Fig. 5. (a) The start and the goal configurations are shown along with the points that cover the contact space. A path that connects the start and the goal is shown in the thick line. The path is extracted from a set of 6 overlapping (b) A 2-d workspace motion planning problem. When the contact space is represented using point set data (shown in the small dark dots), we can efficiently decompose the contact space into star-shaped components, which then can help us finding a path (shown in the thick line) connecting the start and the goal configurations. The horizontal axis in the figure represents the robot's orientational degree of freedom. A^{*} components. (c) The start configuration is shown (left) along with the points that cover the contact space (right).

3-d free-flying robot. The motion planning problem shown in Fig. 5(c) has a long robot trapped inside a hole of the obstacle. In this problem, the robot needs to move out of the hole and has 6 degrees of freedom. Fig. 5(c) also shows a “slice” of the contact space. We generate 825,000 configurations in total to cover the contact space and only 103 guards are needed to cover the entire point-based contact space. In this experiment, we use $k = 10$ in the k -means clustering method for each guard during the visibility computation.

Compare to Visibility PRM. We also use the visibility PRM [18] to solve these three problems above. With 100 runs of the visibility PRM for each problem, on average, 215 guards are created. On the contrary, A^{*} decomposition approaches generate only 145 guards. This is not surprising because the visibility PRM randomly selects the positions of the guards, while our methods attempt to improve the guards by computing visibility kernels and iteratively move guards to increase their visible ranges. However, because of the same reason, the visibility PRM is several times faster than our methods.

TABLE II
EXPERIMENTAL RESULTS

		point generation	A^{*} decomp.	total
Fig. 5(a)	time (sec)	0.27	0.31	0.58
	size	1,327 points	8 guards	
Fig. 5(b)	time (sec)	4.17	40.70	44.87
	size	8,947 points	34 guards	
Fig. 5(c)	time (sec)	478.5	86.2	564.7
	size	825,000 points	103 guards	

VIII. CONCLUSION AND FUTURE WORK

In this paper, we present two motion planning methods using the approximate star-shaped decomposition of the contact space represented by point set data. These two methods are deterministic and probabilistic extensions of the framework in Algorithm I.1. The resulting representation of the free configuration space is composed of a set of guards. These guards collectively can see the entire contact space and they can be connected into a network to capture the connectivity of the free space. Although the proposed methods are promising, it is not without limitations. For

example, the proposed methods is not efficient. It takes more time than regular PRM methods.

We view A^{*} -based motion planner as a complement to the PRM methods and the traditional deterministic approaches. It provides a deterministic approach while can still be applied to high dimensional problems. One of the main challenges is to extend our methods to handle articulated robots.

REFERENCES

- [1] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.
- [2] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.
- [3] F. Lingelbach, “Path planning using probabilistic cell decomposition,” in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2004, pp. 467–472.
- [4] G. Varadhan and D. Manocha, “Star-shaped roadmaps: A deterministic sampling approach for complete motion planning,” in *Proc. Robotics: Sci. Sys. (RSS)*, 2005.
- [5] S. M. LaValle, *Planning Algorithms*, 6th ed. Cambridge University Press, 2006.
- [6] J. M. Keil, “Polygon decomposition,” in *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Amsterdam: Elsevier Science Publishers B.V. North-Holland, 2000, pp. 491–518.
- [7] —, “Decomposing polygons into simpler components,” Ph.D. dissertation, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, 1983.
- [8] S. K. Ghosh, “A linear time algorithm for decomposing a monotone polygon into star-shaped polygons,” in *Proc. 3rd Conf. Found. Softw. Tech. Theoret. Comput. Sci.*, 1983, pp. 505–519.
- [9] V. Chvátal, “A combinatorial theorem in plane geometry,” *J. Combin. Theory Ser. B*, vol. 18, pp. 39–41, 1975.
- [10] J. O’Rourke and K. J. Supowit, “Some NP-hard polygon decomposition problems,” *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 181–190, 1983.
- [11] D. Lee and A. Lin, “Computational complexity of art gallery problems,” *IEEE Trans. Inform. Theory*, vol. 32, no. 2, pp. 276–282, 1986.
- [12] T. K. Dey, J. Giesen, and S. Goswami, “Shape segmentation and matching with flow discretization,” in *Proc. Workshop on Algorithms and Data Structures*, 2003, pp. 25–36.
- [13] I. Yamazaki, V. Natarajan, Z. Bai, and B. Hamann, “Segmenting point sets,” in *IEEE Intl. Conf. Shape Modeling and Applications (SMI)*, 2006, pp. 4–13.
- [14] J.-M. Lien, “Approximate star-shaped decomposition of point set data,” in *Proceedings of the IEEE/Eurographics Symposium on Point Based Graphics (PBG)*, 2007.
- [15] —, “Point-based minkowski sum boundary,” in *PG ’07: Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 261–270.
- [16] A. Kaul and J. Rossignac, “Solid-interpolating deformations: construction and animation of PIPs,” in *Proc. Eurographics ’91*, 1991, pp. 493–505.
- [17] L. E. Kavvaki, P. Svestka, J. C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.
- [18] C. Nissoux, T. Simeon, and J.-P. Laumond, “Visibility based probabilistic roadmaps,” in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 1999, pp. 1316–1321.