Article

The International Journal of
Robotics Research
2014, Vol. 33(14) 1711–1725
© The Author(s) 2014
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0278364914556517
ijr.sagepub.com

Sparsification of motion-planning roadmaps by edge contraction

Oren Salzman[1], Doron Shaharabani[1], Pankaj K. Agarwal[2] and Dan Halperin[1]

Abstract

Roadmaps constructed by the recently introduced PRM algorithm for asymptotically-optimal motion planning encode high-quality paths yet can be extremely dense. We consider the problem of* sparsifying *the roadmap, i.e. reducing its size, while minimizing the degradation of the quality of paths that can be extracted from the resulting roadmap. We present a simple, effective sparsifying algorithm, called* roadmap sparsification by edge contraction *(RSEC). The primitive operation used by RSEC is* edge contraction *—the contraction of a roadmap edge (v′,v″) to a new vertex v′ and the connection of the new vertex* v *to the neighboring vertices of the contracted edge's vertices (i.e. to all neighbors of* v′ *and* v″*). For certain scenarios, we compress more than 97% of the edges and vertices of a given roadmap at the cost of degradation of average shortest path length by at most 4%.*

Keywords

Motion control, mechanics, design and control, manipulation planning, manipulation, path planning for manipulators

1. Introduction

Motion planning is a fundamental research topic in robotics with applications in diverse domains such as graphical animation, surgical planning, computational biology, computer games and of course robot motion. Sampling-based planners (Kavraki et al., 1996; Hsu et al., 1999; Kuffner and Lavalle, 2000), introduced in the 1990s, enabled the solving of motion-planning problems that had been previously infeasible (Choset et al., 2005). Specifically, for multi-query scenarios, planners such as the Probabilistic Roadmap Planner (PRM) (Kavraki et al., 1996) approximate the connectivity of the free space ($\mathcal{C}_{\text{free}}$) by taking random samples from the configuration space (C-space) and connecting nearby free configurations when possible. The resulting data structure, the *roadmap*, is a graph in which vertices represent configurations in $\mathcal{C}_{\text{free}}$ and edges are collision-free paths connecting two such configurations.

Answering a motion-planning query using such a roadmap is subdivided into (i) connecting the source and target configurations of the query to the roadmap and (ii) finding a path in the roadmap between the connection points. Thus, a roadmap should cover the C-space (*coverage* property) and be connected when the C-space is connected (*connectivity* property). Typically, a path of *high quality* is desired, where quality can be measured in terms of, for example, length, clearance, smoothness, energy, to mention a few criteria, or some combination of the above. As the connectivity property alone does not guarantee high-quality paths (Nechushtan et al., 2010; Karaman and Frazzoli, 2011), additional work is required.

Motion-planning algorithms that create high-quality roadmaps, however, result in large, dense graphs (Schmitzberger et al., 2002; Nieuwenhuisen and Overmars, 2004; Jaillet and Siméon, 2006; Karaman and Frazzoli, 2011; Raveh et al., 2011). This may be undesirable due to prohibitive storage requirements and long online query processing times. We seek a compact representation of the roadmap graph without sacrificing path quality.

1.1. Related work

Geraerts and Overmars (2007) suggested an algorithm for creating small roadmaps of high-quality paths (namely *short* and with *high clearance*). An alternative approach to creating small roadmaps that allows for high-clearance paths is the work by Lien et al. (2003). Siméon et al.

[1]Blavatnic School of Computer Science, Tel-Aviv University, Israel
[2]Department of Computer Science, Duke University, USA

Corresponding author:
Oren Salzman, Blavatnic School of Computer Science, Tel-Aviv University PO Box 39040, Tel-Aviv 69978, Israel.
Email: orenzalz@post.tau.ac.il

(2000) exploited the structure of the configuration space to produce small, high-quality roadmaps using the notion of visibility between sampled configurations. The work on graph spanners (Peleg and Schäffer, 1989; Narasimhan and Smid, 2007) is closely related to our problem of computing a compact representation of a roadmap graph. Formally, a t-spanner of a graph $G = (V, E)$ is a sparse subgraph $G' = (V, E')$, with $E' \subseteq E$ such that the shortest path between any pair of points in G' is no longer than t times the shortest path between them in G. The parameter t is referred to as the *stretch* of the spanner. It is well known that small size $(1 + \varepsilon)$-spanners exist for complete Euclidean graphs (Arya et al., 1995; Narasimhan and Smid, 2007; Elkin and Solomon, 2013). Spanners have been successfully used to compute collision-free approximate shortest paths amid obstacles in 2D and 3D or to compute them on a surface (Clarkson, 1987; Har-Peled, 1999; Varadarajan and Agarwal, 2000; Aleksandrov et al., 2005). In the context of roadmap constructions, Marble and Bekris (2011) introduced the notion of *asymptotically near-optimal roadmaps*—roadmaps that are guaranteed to return a path whose quality is within a guaranteed factor of the optimal path. They apply a graph-spanner algorithm to an existing roadmap to reduce its size (we call their algorithm SPANNER). Specifically, they use the spanner algorithm by Baswana and Sen which, given an integer k and a graph, returns an optimal $(2k - 1)$-spanner in time linear in the number of edges times k. The algorithm works by constructing clusters of nodes and then connecting nearby clusters. Recently, graph-spanner algorithms have also been incorporated in the construction phase of the roadmap itself (Dobson et al., 2012; Marble and Bekris, 2012; Dobson and Bekris, 2013; Marble and Bekris, 2013; Wang et al., 2013).

A drawback of the spanner-based approach is that it only reduces the number of edges of the graph and does not remove any of its vertices. In the context of roadmaps, it is desirable to remove redundant vertices as well, and to construct a small-size graph in the free space. Such an approach has recently been proposed for computing approximate shortest paths for a point robot amid convex obstacles in two or three dimensions (Agarwal et al., 2009), but it is not clear how to extend this approach to higher dimensional configuration spaces.

The problem of computing a compact representation of a roadmap graph falls under the broad area of computing data summaries or computing a hierarchical representation of data. There has been extensive work in this area in the last decade because of the need to cope with big data sets (Agarwal et al., 2005; Cormode et al., 2012). The work in computer graphics on computing a hierarchical representation of a surface, represented as a triangulated mesh, is perhaps the most closely related work to our approach (Luebke et al., 2002). The surface-simplification problem asks for simplification of the mesh—a reduction in size—while ensuring that the resulting surface approximates the original one within a prescribed error tolerance (Luebke et al., 2002). Some versions of the optimal surface-simplification problem, the problem of computing a smallest-size surface, are known to be NP-Hard (Agarwal and Suri, 1998), and several approximation algorithms and heuristics have been proposed (Garland and Heckbert, 1997; Edelsbrunner, 2001). The practical algorithms progressively simplify the surface by modifying its topology locally at each step, e.g. removing a vertex and retriangulating the surface, or contracting an edge. The *edge-contraction* method has now become the most popular method for simplifying a surface (Hoppe et al., 1993; Garland and Heckbert, 1997; Edelsbrunner, 2001).

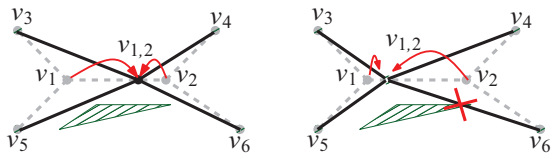
We adapt the widely used *edge-contraction* technique for surface simplification to roadmap simplification. We present a simple, effective algorithm, called *roadmap sparsification by edge contraction* (RSEC), to sparsify a roadmap, in which edge contraction is the primitive operation. Our algorithm often generates very sparse subgraphs while exhibiting little degradation in path quality when compared to the original graph. In contrast to the algorithm of Marble and Bekris (2011) in which the set of vertices remains intact, RSEC dramatically reduces the number of vertices.

We note that RSEC is an offline algorithm that is run *after* the generation of a highly dense roadmap. Applying a sparsification algorithm in the construction stage of the roadmap (Dobson et al., 2012; Marble and Bekris, 2012; Dobson and Bekris, 2013; Marble and Bekris, 2013) would be more desirable and we plan to explore this as part of our future work.

Section 2 describes the overall algorithmic framework of RSEC, Section 3 describes the criteria we use to contract an edge, Section 4 provides various details of the algorithm and Section 5 presents experimental results comparing alternative choices made by our algorithm. For certain scenarios, we compress more than 97% of the edges and vertices while causing degradation of average path length by at most 4%. Additionally, we compare RSEC with the algorithm presented by Marble and Bekris (2011). We show that our algorithm produces paths of higher quality than theirs when applying the same compression rate while also being able to surpass the maximum compression rate achieved by their algorithm. We conclude in Section 6 with suggestions for further research.

2. Algorithmic framework

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph, output of a PRM-type algorithm approximating $\mathcal{C}_{\text{free}}$, where $\mathcal{V} \subseteq \mathcal{C}_{\text{free}}$ is a set of points (configurations) in the free space, and each edge $(u, v) \in \mathcal{E}$ is a line segment $uv \subset \mathcal{C}_{\text{free}}$ connecting the points $u, v \in \mathcal{V}$. We wish to construct a graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ which is a compact approximation of $\mathcal{C}_{\text{free}}$ while maintaining the coverage of the roadmap and the quality of the approximation provided by $\mathcal{G}$. In this paper we concentrate on the *length* of a path as its quality measure.[1] We begin by introducing some notation, then describe the primitive



(a) Legal edge contraction. (b) Illegal edge contraction.

Fig. 1. Examples of an edge contraction: the edge (v_1, v_2) is contracted to the point $v_{1,2}$. Original edges are drawn in gray, new edges are drawn in black and an obstacle is drawn in green.

edge contraction and some of its properties, and finally give an overview of the algorithm for constructing the sparse graph $\mathcal{G}'$.

Let $\mathcal{H} = (V, E)$ be an arbitrary undirected graph whose vertices are points in $\mathcal{C}_{\text{free}}$ and edges are line segments connecting their endpoint such that each line segment lies inside $\mathcal{C}_{\text{free}}$. For an edge $e \in E$, we denote its (Euclidean) length by $\|e\| = \|uv\|$. Similarly for a path $P = \langle v_1, \ldots, v_k \rangle$ in $\mathcal{H}$, we denote its length by $||P|| = \sum_{i=1}^{k-1} \|v_i v_{i+1}\|$. For a vertex $v \in V$, the set of *neighboring vertices* of v is defined as $\text{nbr}(v) = \{u | (u,v) \in E\}$. For an edge $(u, v) \in E$, we define its neighboring vertices to be $\text{nbr}(u, v) = \text{nbr}(u) \cup \text{nbr}(v) \setminus \{u,v\}$.

2.1. Edge contraction

Let $\mathcal{H}$ be the same as above. Given an edge $(u, v) \in E$ and a point $p \in \mathcal{C}_{\text{free}}$, an *edge contraction* of (u, v) to p, denoted $(u, v; p)$, is the following process:

(i) Add p as a new vertex of $\mathcal{H}$,
(ii) add an edge (w, p) for each vertex $w \in \text{nbr}(u, v)$, and
(iii) remove from $\mathcal{H}$ the vertices u, v and all the edges incident on them.

We say that (u, v) is *contracted* to p, and that p is the *contraction point* of (u, v). The edge contraction $(u, v; p)$ is called *legal* if all new edges, i.e. the ones incident on p lie in $\mathcal{C}_{\text{free}}$. For a visualization of legal and illegal edge contractions, see Fig 1. The vertices u, v are called the *parents* of p.

Suppose we perform a sequence of edge contractions: $(u_1, v_1; p_1), \ldots, (u_k, v_k; p_k)$. Let G_0 be the initial graph $\mathcal{G}$, and let $G_i = (V_i, E_i)$ be the graph obtained from G_{i-1} by performing the edge contraction $(u_i, v_i; p_i)$. A vertex of $u \in G_j$ is an *ancestor* of a vertex $v \in G_i$, for $i \geq j$, if either $u = v$ or u is an ancestor of a parent of v. Let $\text{anc}(v)$ denote the set of ancestors of v. The following properties of graphs G_i are straightforward:

(P1) For each original vertex $v \in \mathcal{V}$, there exists exactly one vertex $u \in V_k$ such that $v \in \text{anc}(u)$; we set $u = \varphi(v)$.
(P2) Let $(u, v) \in \mathcal{E}$ be an original edge in $\mathcal{G}$. G_k either contains a vertex $w \in V_k$ such that $u, v \in \text{anc}(w)$ or an edge $(w, z) \in E_k$ such that $u \in \text{anc}(w)$ and $v \in \text{anc}(z)$.

Algorithm 1. RSEC (G).

```
1: G' ← G
2: Q ← initialize_queue(E)
3: while not_empty(Q) do
4:    e ← Q.pop_head
5:    p ← get_contraction_point(e)
6:    if p ≠ NIL then
7:       if edge_contraction (G', e, p) = SUCCESS then
8:          Q ← update_queue()
```

(P3) Let $(w, z) \in E_k$ be an edge in G_k. There exists an edge $(u, v) \in \mathcal{E}$ in $\mathcal{G}$ such that $u \in \text{anc}(w)$ and $v \in \text{anc}(z)$.

2.2. RSEC algorithm

Before describing the RSEC algorithm, we remark that we wish to preserve the quality and connectivity of the original input graph, so we add the constraint that vertices in the sparsified graph $\mathcal{G}'$ should not be too far from their ancestors. Thus, given a *drift bound* $\delta > 0$, we require the algorithm to maintain the following invariant:

Bounded-drift invariant : $\forall v \in \mathcal{V}', \ \forall u \in \text{anc}(v) \ ||uv|| \leq \delta$.

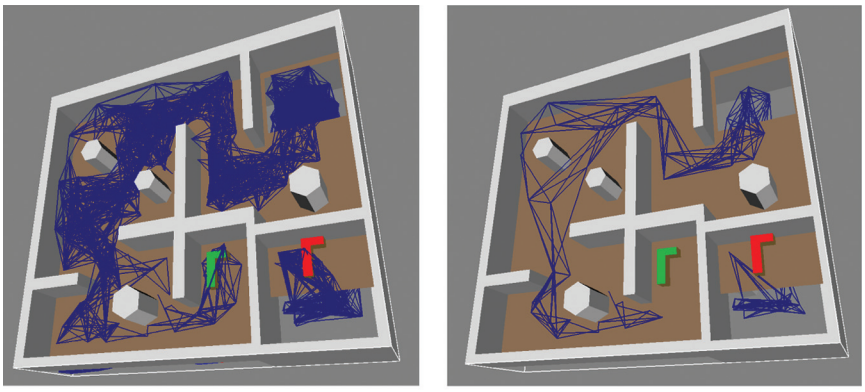
In practice, we normalize the drift bound to be δ/a where a is the length of the diagonal of the bounding box of the workspace. In the rest of this paper, we use the term *drift bound* to refer to the normalized drift bound.

We are now ready to describe the RSEC algorithm: The algorithm performs a sequence of legal edge contractions, as outlined in Algorithm 1. It assigns a *priority* to each candidate edge for contraction (see Section 4 for details) and stores the candidate edges in a priority queue. At each step, an edge $e = (u, v)$ with the highest priority is removed from the priority queue, and the procedure `get_contraction_point(e)`, described in Section 4, is executed. At a high level, this procedure returns a point $p \in uv$, if there exists one, that satisfies the bounded-drift invariant and locally minimizes the overall 'degradation' of paths.

Next, RSEC calls the procedure `edge_contraction`, outlined in Algorithm 2, which first checks whether the edge contraction $(e; p)$ is legal, using a local planner.[2] If so, it performs the edge contraction $(e; p)$: contracts the

Algorithm 2. edge_contraction $(H = (E, V), (u, v), p)$.

```
1: for all w ∈ H.nbr(v) ∪ H.nbr(u) do
2:    if local_planner(w, p) = FAILURE then
3:       return FAILURE
4: for all v' ∈ H.nbr(v) do
5:    E ← E ∪ {(v', p)} \ {(v', v)}
6: for all u' ∈ H.nbr(u) do
7:    E ← E ∪ {(u', p)} \ {(u', u)}
8: V ← V ∪ {p} \ {v, u}
9: return SUCCESS
```



(a) Original roadmap. **(b)** Sparsified roadmap.

Fig. 2. Example of a roadmap before and after running RSEC on the Cubicles scenario.

edge e to the vertex p, removes the edges incident on u and v, and adds the edge wp for each $w \in \text{nbr}(u, v)$.

Finally, the algorithm computes the priority of the new edges and adds them to the priority queue.

We note that contracting an edge (u, v) in a graph $\mathcal{G}$ reduces its size—the number of vertices decreases by one and the number of edges decreases by at least one (the edge (u, v)). Additionally, for every common neighbor w of u and v, the edges (u, w) and (v, w) merge into a single edge (p, w). Figure 2 depicts an original roadmap and the sparsified roadmap after running the RSEC algorithm. The original graph has 1000 vertices and 14,450 edges. The algorithm performs a sequence of 886 edge contractions, and the final graph has 116 vertices and 535 edges.

3. Utility function

In this section we describe a utility function, called the *degradation factor*, which attempts to capture the effect of edge contraction on the length of paths. It is used to choose the contraction point of an edge and to order the edges in the priority queue (see Section 4). We begin by describing the effect of an edge contraction on a path, then use this to define the degradation factor, and finally bound the degradation of the length of a path incurred by a sequence of edge contractions.

3.1. Edge contraction and paths

Let P be a path before an edge (u, v) is contracted to a point p. If neither u nor v lies on P, then P is unaffected by this edge contraction. If P contains the edge (u, v), then it shrinks to the point p in P and the path becomes shorter (e.g. P_1 in Figure 3). If P contains both u and v but not the edge (u, v), i.e. they are not adjacent in P, then after the edge contraction P contains a cycle, starting and ending at p. We can remove this cycle and obtain a shorter path. Finally, if only one of u and v, say u, appears on P, then the edges (w, u) and (u, z) adjacent to u in P become (w, p) and (p, z), respectively, and the new edges may be longer than the original ones. So P may become longer in this case (e.g. P_2 in Figure 3).

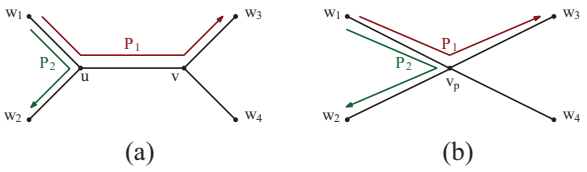


Fig. 3. The effect on paths of the edge contraction $(u, v; v_p)$. The graph (a) before and (b) after the contraction of the edge (u, v). We consider two paths P_1 and P_2. P_1 originally passes through both u and v, and is shorter after the edge is contracted. P_2 originally passes through u but not through v, and is longer after the edge contraction.

For a vertex v in the original graph $\mathcal{G}$, let $\varphi(v)$ be the corresponding vertex in $\mathcal{G}'$, as defined in property (P1) of edge contractions. We map a path $P = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k in the original graph $\mathcal{G}$ to a path $\varphi(P)$ from $\varphi(v_1)$ to $\varphi(v_k)$ in the sparsified graph $\mathcal{G}'$, recursively as follows: if $\varphi(v_1) = \varphi(v_k)$, then $\varphi(P)$ consists of a single vertex $\varphi(v_1)$. Otherwise, let $r < k$ be the largest index such that $\varphi(v_1) = \varphi(v_r)$, i.e. $\varphi(v_i) \neq \varphi(v_1)$ for all $r < i \leq k$, and let $P' = \langle v_{r+1}, \ldots, v_k \rangle$ be the suffix of P. Suppose $\varphi(v_1) = \varphi(v_r) = v'_1$ and $\varphi(P') = \langle v'_2, \ldots, v'_{k'} \rangle$, then $\varphi(P) = \langle v'_1, v'_2, \ldots, v'_{k'} \rangle$.

Since $v'_2 = \varphi(v_{r+1})$, $\varphi(v_r) \neq \varphi(v_{r+1})$, and $(v_r, v_{r+1}) \in \mathcal{E}$, then, by property (P2) of edge contraction $(v'_1, v'_2) \in \mathcal{E}'$. Hence, $\varphi(P)$ is a valid path in $\mathcal{G}'$. We will bound the length of $\varphi(P)$ in terms of the length of P.

3.2. Degradation factor

We now define the degradation factor, which bounds the increase in length of a path due to an edge contraction. As the endpoints of a contracted edge may themselves be

contracted edges, we maintain this change relative to the *original* vertices and give a recursive definition.

Definition 1 (Degradation factor). The ***degradation factor*** for an edge (u, v) in the sparsified graph, denoted by $\eta(u, v)$, is defined as follows: for each edge (w, z) of the original graph $\eta(w, z) = 1$. For a new edge (w, p) created by the contraction of the edge (u, v) to the point p

$$\eta(w,p) = \begin{cases} \eta(w,u)\frac{||wp||}{||wu||} & w \in \mathrm{nbr}(u),\ \setminus\ \mathrm{nbr}(v) \\ \eta(w,v)\frac{||wp||}{||wv||} & w \in \mathrm{nbr}(v) \setminus \mathrm{nbr}(u) \\ \max\left\{\eta(w,u)\frac{||wp||}{||wu||}, \eta(w,v)\frac{||wp||}{||wv||}\right\} & w \in \mathrm{nbr}(u) \cap \mathrm{nbr}(v) \end{cases} \quad (1)$$

By property (P2), each edge (u, v) in the original graph $\mathcal{G}$ has a corresponding edge $(\varphi(u), \varphi(v))$ in the sparsified graph $\mathcal{G}'$ if $\varphi(u) \neq \varphi(v)$. The following lemma bounds the length of $(\varphi(u), \varphi(v))$ using the degradation factor.

Lemma 3.1. *Let* $(u', v') \in \mathcal{E}'$ *be an edge in* $\mathcal{G}'$. *For every-*$(u, v) \in \mathcal{E}$ *such that* $u \in \mathrm{anc}(u')$, $v \in \mathrm{anc}(v')$, i.e. $u' = \varphi(u)$ *and* $v' = \varphi(v)$

$$||u'v'|| \leq \eta(u', v')||uv||$$

Proof. We prove the lemma by induction on the number of contraction operations that involve u, v, or their descendants. The base case of the induction holds trivially because if $u = u'$ and $v = v'$, i.e. $(u', v') \in \mathcal{E}$, then $\eta(u',v') = 1$ and $||u'v'|| = ||uv||$.

For the induction step, assume that an edge (w', w'') is contracted to the vertex v' and $u' \in \mathrm{nbr}(w', w'')$. Without loss of generality assume that $v \in \mathrm{anc}(w')$ and recall that $u \in \mathrm{anc}(u')$.

If $u' \notin \mathrm{nbr}(w') \cap \mathrm{nbr}(w'')$ then $u' \in \mathrm{nbr}(w') \setminus \mathrm{nbr}(w'')$ By definition, $\eta(u', v') = \eta(u', w')\frac{||u'v'||}{||u'w'||}$, therefore

$$\begin{aligned} ||u'v'|| &= \frac{\eta(u', v')}{\eta(u', w')}||u'w'|| \leq \frac{\eta(u', v')}{\eta(u', w')}\eta(u', w')||uv|| \\ &= \eta(u'v')||uv|| \end{aligned}$$

where the inequality follows from the induction hypothesis.

Similarly if $u' \in \mathrm{nbr}(w') \cap \mathrm{nbr}(w'')$ then

$$\begin{aligned} ||u'v'|| &= \eta(u', v')\max\left\{\frac{||u'w'||}{\eta(u', w')}, \frac{||u'w''||}{\eta(u', w'')}\right\} \\ &\leq \eta(u', v')\max\left\{\frac{\eta(u', w')||uv||}{\eta(u', w')}, \frac{\eta(u', w'')||uv||}{\eta(u', w'')}\right\} \\ &= \eta(u', v')||uv|| \end{aligned}$$

Remark. The bounded-drift invariant guarantees that $||u'v'|| \leq ||uv|| + 2\delta$, but for typical choices of δ, $\eta(u', v') \leq 1 + \frac{2\delta}{||uv||}$ for most edges, so the above lemma gives a sharper bound on $||u'v'||$.

3.3. Bounding path degradation

For a pair of vertices $u, v \in \mathcal{V}$, let $\mathrm{d}_{\mathcal{G}}(u, v)$ be the length of the shortest path between u and v in $\mathcal{G}$, and similarly define $\mathrm{d}_{\mathcal{G}'}(\cdot, \cdot)$ for $\mathcal{G}'$. The following lemma bounds the increase in the length of the shortest path between u and v. We define

$$\eta^* = \max_{e \in \mathcal{E}'} \eta(e)$$

Lemma 3.2. *For any pair* $u, v \in \mathcal{V}$, $\mathrm{d}_{\mathcal{G}'}(\varphi(u), \varphi(v)) \leq \eta^* \mathrm{d}_{\mathcal{G}}(u, v)$.

Proof. Let $P = \langle u = w_1, w_2, \ldots, w_k = v\rangle$ be the shortest path in $\mathcal{G}$ from u to v, and let $\varphi(P) = \langle v'_1, \ldots, v'_{k'}\rangle$ for $k' \leq k$. Then $\mathrm{d}_{\mathcal{G}'}(\varphi(u), \varphi(v)) \leq \|\varphi(P)\| = \sum_{i=1}^{k'} \left\|v'_i v'_{i+1}\right\|$. For each $i < k'$, let $j_i \leq k$ be the largest index in P such that $v_{ji} \in \mathrm{anc}(v'_i)$, i.e. $v'_i = \varphi(v_{ji})$ and $\varphi(v_l) \neq v'_i$ for all $j_i < l \leq k$. By property (P3), $(v_{j_i}, v_{j_i+1}) \in \mathcal{E}$, therefore by Lemma 3.1

$$\left\|v'_i v'_{i+1}\right\| \leq \eta(v'_i, v'_{i+1})\left\|v_{j_i} v_{j_i+1}\right\| \leq \eta^*\left\|v_{j_i} v_{j_i+1}\right\|$$

Hence

$$\|\varphi(P)\| = \sum_{i=1}^{k'} \left\|v'_i v'_{i+1}\right\| \leq \eta^* \sum_{i=1}^{k'} \left\|v_{j_i} v_{j_i+1}\right\| \leq \eta^* \|P\|$$

For a pair of vertices $u, v \in \mathcal{V}$, suppose we connect u to all those vertices $u' \in \mathcal{V}'$ such that $\|uu'\| \leq \delta$ and do the same for v. Let $\overline{\mathrm{d}}_{\mathcal{G}'}(u, v)$ be the length of the shortest path in the resulting graph. Then the following bound on $\overline{\mathrm{d}}_{\mathcal{G}'}(u, v)$ follows from the above lemma.

Corollary 3.3. *For any pair* $u, v \in \mathcal{V}$, $\overline{\mathrm{d}}_{\mathcal{G}'}(u, v) \leq \eta^* \mathrm{d}_{\mathcal{G}}(u, v) + 2\delta$.

4. Algorithmic details

With the definition of the degradation factor at hand, we can now describe the missing details of the RSEC algorithm. Specifically, how do we choose the contraction point of an edge, and how do we order the edges in our queue to obtain a smaller graph of high quality?

4.1. Choice of contraction point

In order to minimize the path degradation, we choose the contraction point of an edge (u, v) that maintains the bounded-drift invariant and that minimizes the degradation factor of the neighboring edges of (u, v).

A point p on a segment uv can be parametrized as

$$p(\alpha) = u + \alpha(v - u), \quad \alpha \in [0, 1]$$

For a point $x \in \mathcal{C}_{\mathrm{free}}$, the set of values of α that satisfy the constraint $\|xp(\alpha)\| \leq \delta$ forms a contiguous interval I_x. Set $\mathcal{I}_{uv} = \cap I_x$, for $x \in \mathrm{anc}(u) \cup \mathrm{anc}(v)$. For any $\alpha \in \mathcal{I}_{uv}$,

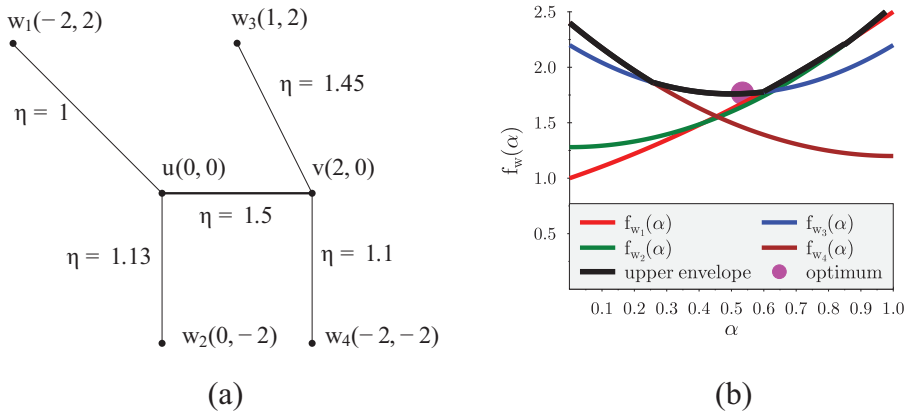


Fig. 4. (a) An edge (u, v) (in bold), which is considered for contraction and its neighboring edges. (b) Upper envelope of parabolic arcs (marked in black) representing $f_w(\alpha)$ for all neighboring vertices w of u and v. The contraction point that minimizes the maximal value is the optimal local degradation factor.

$p(\alpha)$ satisfies the bounded-drift invariant. Our goal is to choose a value $\alpha^* \in \mathcal{I}_{uv}$ that minimizes the degradation factor of the neighboring edges of (u, v).

For a vertex $w \in \text{nbr}(u)$, we define the function $f_w : [0, 1] \to \mathbb{R}_{\geq 0}$, where $f_w(\alpha)$ is the square of the degradation factor of the edge (w, u) as its endpoint u is moved to the contracted point $p(\alpha)$ of (u, v). More precisely

$$f_w(\alpha) = \eta^2(u,w)\frac{\|wp(\alpha)\|^2}{\|uw\|^2} = \frac{\eta^2(u,w)}{\|uw\|^2}\|(u-w)+\alpha(v-u)\|^2$$
$$= \frac{\eta^2(u,w)}{\|uw\|^2}\left[\|uw\|^2 + 2\alpha\langle u-w, v-u\rangle + \alpha^2\|vu\|^2\right] \quad (2)$$

Similarly for a vertex $w \in \text{nbr}(v)$, we define the function f_w as

$$f_w(\alpha) = \eta^2(v,w)\frac{\|wp(\alpha)\|^2}{\|vw\|^2} = \frac{\eta^2(v,w)}{\|vw\|^2}$$
$$\left[\|uw\|^2 + 2\alpha\langle u-w, v-u\rangle + \alpha^2\|vu\|^2\right] \quad (3)$$

Using the functions f_w, for $w \in \text{nbr}(u, v)$, we propose the following two different criteria to choose α^*.

(i) Minimize the ***maximum*** degradation factor of the neighboring edges. Define the function

$$F^{uv}_{\infty}(\alpha) = \max\left\{\max_{w\in\text{nbr}(u)} f_w(\alpha), \max_{z\in\text{nbr}(v)} f_z(\alpha)\right\}$$

and set

$$\alpha^*_{\infty} = \arg\min_{\alpha\in\mathcal{I}_{uv}} F^{uv}_{\infty}(\alpha) \quad \text{and} \quad \text{err}_{\infty}(u,v) = F^{uv}(\alpha^*_{\infty})$$

That is, $p(\alpha^*_{\infty})$ is the point on the segment uv that satisfies the bounded-drift invariant and minimizes the maximum degradation factor of the neighboring edges.

Geometrically, the graph of each function f_w is a parabola, F^{uv}_{∞} is the upper envelope of a set of parabolic functions, and α^*_{∞} is the value of $\alpha \in I_{uv}$ corresponding to the lowest point on the upper envelope. See Figure 4. F^{uv}_{∞} is a convex function with, at most, $2\kappa_{uv}$ breakpoints (Sharir and Agarwal, 1995), where $\kappa_{uv} = |\text{nbr}(u)| + |\text{nbr}(v)|$. The lowest point on F^{uv}_{∞}, which might not lie inside the range I_{uv}, is either a breakpoint of F^{uv}_{∞} or the lowest point of one of the parabolas. Therefore F^{uv}_{∞} and α^*_{∞} can be computed in $O(\kappa_{uv})$ time.

A drawback of choosing α^*_{∞} is that it is biased toward outliers, so we propose a different criterion which will be the one used by RSEC.

(ii) Minimize the ***sum of squares of*** degradation factors of the neighboring edges. We now define the function

$$S^{uv}(\alpha) = \sum_{w\in\text{nbr}(u)} f_w(\alpha) + \sum_{z\in\text{nbr}(v)} f_z(\alpha)$$

and set

$$\alpha^* = \arg\min_{\alpha\in\mathcal{I}_{uv}} S^{uv}(\alpha) \quad \text{and} \quad \text{err}(u,v) = S^{uv}(\alpha^*) \quad (4)$$

The graph of S^{uv} is a parabola, and $S^{uv}(\alpha^*)$ is the lowest point on this parabola inside the range I_{uv}, which can be computed in $O(1)$ time.

We remark that $S^{uv}(\alpha)$ is not the same as the sum of squares of degradation factors of the neighboring edges of (u, v) because for a vertex $w \in \text{nbr}(u) \cap \text{nbr}(v)$, we take the sum of the degradation of the edges (w, u) and (w, v), instead of taking their maximum, as defined in equation (1). We choose this form because it is sufficiently close and the functional form is considerably simpler.

4.2. Edge ordering

We propose two different criteria for choosing the next edge $e = (u, v)$ for contraction, as follows:

(i) At each step we choose the edge (u, v) with the minimum value of κ_{uv}, the sum of degrees of the endpoints of (u, v). We refer to this approach as *deg_sum*. The

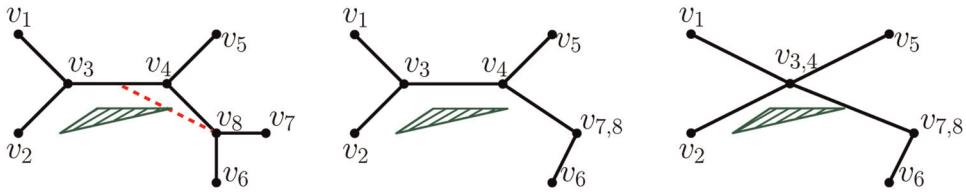


Fig. 5. Scenario where re-inserting an edge to the priority queue enables additional edge contractions. (a) Edge (v_3, v_4) is considered for contraction but the contraction is not valid and the edge is removed from the queue as the prospective edge from v_8 will collide with the green obstacle (demonstrated by the dashed red line). (b) The edge (v_7, v_8) is contracted to the point $v_{7,8}$. (c) Re-inserting the edge (v_3, v_4) to the queue allows it to be contracted to the point $v_{3,4}$.

Table 1. Implementation alternatives as described in this section. Unless stated otherwise, the variant that we use minimizes the sum of squares of the degradation factor for the choice of contraction point, uses df_sum as the edge ordering heuristic, and re-inserts edges into the priority queue.

Choice of contraction point	Edge ordering	Edge re-insertion
Minimizing maximal degradation factor	deg_sum	yes
Minimizing sum of squares of degradation factor	df_sum	no

intuition behind this criterion is that edges with few neighboring edges degrade few paths. Furthermore, checking whether an edge lies inside the free space is expensive, so checking whether an edge contraction is legal is more efficient for edges with a low sum of endpoint degrees.

(ii) At each step we choose the edge (u, v) with the minimum value of err (u, v), defined in equation (4). The intuition behind this approach is that it is likely to minimize the overall degradation of paths. By choosing the sum of squares of degradation factors instead of their average, we bias the choice toward low-degree edges. We refer to this approach as *df_sum*.

An additional issue that should be addressed regards the question "should an edge be re-inserted into the priority queue once it has been determined not to be contractible?" Consider an edge (u, v) that was removed from the queue because it could not be contracted. This implies that a neighbor w of u (or v) may not be connected to the contraction point of (u, v). Once w is moved (due to a different edge contraction), it is possible that (u, v) becomes contractible. An example of such a scenario is depicted in Figure 5. Obviously, re-inserting these edges is costly in runtime but improves the compression achieved. Since the sparsification step is performed offline, we re-insert the edges into the queue. We demonstrate the advantage of this choice in experiments reported in Section 5.

In this section we described several different ways to implement the algorithmic framework presented in Section 2. Table 1 summarizes the alternatives suggested. An experimental comparison of several alternatives is presented in Section 5.

5. Evaluation

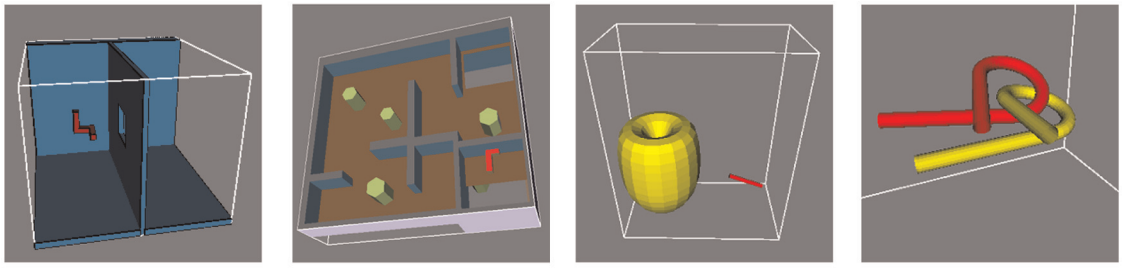
In this section we report on our experimental results. We first evaluate the two edge-ordering approaches described in Section 4 namely *deg_sum* and *df_sum*. Then we discuss the effect of re-inserting edges to the priority queue. Next we evaluate the performance of the RSEC algorithm using the following criteria: connectivity, compression factor, and roadmap quality of the sparsified network. Wherever relevant, we compare the results obtained with RSEC to those obtained with SPANNER. Throughout the section when we say *graph* we refer to the *roadmap graph*.

We begin by describing the experimental setup and show preliminary results that justify our heuristic choices. Next, we report on the main results of RSEC—the connectivity obtained by the algorithm, the compression rates and path quality obtained. Finally, we report on additional insights we have gained regarding the behavior of RSEC.

5.1. Experimental setup and preliminary results

We evaluated our algorithm on four scenarios provided by the OMPL distribution (Şucan et al., 2012), and depicted in Figure 6. The Easy and Bug trap scenarios' C-space has two different large components connected by a small passage, the Cubicles scenario's C-space has several homotopy classes due to the scattered obstacles, and the Alpha puzzle has a narrow passage in its C-space. We ran the PRM* (Karaman and Frazzoli, 2011) algorithm to obtain two different initial roadmaps. The first, which we call *normal setting*, is a roadmap with 5000 vertices and approximately 100,000 edges. The second, which we call *dense setting*, has 20,000 vertices and approximately 1.2 million edges. The dense setting resembles the benchmark used by Marble and Bekris (2011). The crucial difference between our two settings is not the size of the graph but the average degree of each vertex: 40 and 120 for the normal setting and dense setting, respectively.

We measure the quality of a roadmap in terms of path length and report two values: (i) *path degradation* and (ii) *compression factor*. Path degradation is measured by



(a) Easy. **(b)** Cubicles. **(c)** Bug trap. **(d)** Alpha puzzle.

Fig. 6. Environments used for the experiments. The robot is depicted in red. All scenarios were taken from the OMPL (Şucan et al., 2012) distribution.

picking 1000 random pairs of start and goal configurations and performing a shortest-path query for each pair using the original roadmap and the compressed roadmap. Let $||P||$ denote the length of the path obtained using the original roadmap and $||P'||$ denote the length of the path obtained using the compressed roadmap, then the path degradation is the ratio $\frac{||P'||}{||P||}$. The compression factor is the ratio $\frac{|\mathcal{G}|}{|\mathcal{G}'|}$, where $|\mathcal{G}|$ is the size of the original roadmap and $|\mathcal{G}'|$ is the size of the sparsified roadmap. We assume that each vertex stores the coordinates of the configuration it represents (which is proportional in size to the number of degrees of freedom) and each edge stores the indices of the vertices it connects and its weight (namely the degradation factor). Thus for our scenarios, where the robot has six degrees of freedom, we measure the size of a roadmap $\mathcal{G}(V, E)$ as $|\mathcal{G}| = 6|V| + 3|E|$.

All experiments were run using the Open Motion Planning Library (OMPL 0.10.2) (Şucan et al., 2012) on a 3.4 GHz Intel Core i7 processor with 8 GB of memory. All results are averaged over at least five different runs. For each experiment described, if similar results are observed in all scenarios we only give a representative sample. In many plots, we report on the path degradation for the same compression factor for different experiments. As we have error bars, this can become quite cluttered. Thus, for brevity, we slightly shift the results along the x-axis for different experiments on the same plot. Additionally we note that we use the OMPL (Şucan et al., 2012) definition of path length for our C-spaces, which uses a weighted combination between the Euclidean distance and angular distance; see Şucan et al. (2012) for details. We conclude this subsection by presenting experimental results that justify the heuristic choices we made. In all our experiments we use a maximal drift value of 0.16. This seems to be the value that leads to the highest compression rates while still keeping the largest path degradation (i.e. outliers) within an acceptable factor from the average path degradation (see Section 5.3 for more on the subject).

5.1.1. Evaluation of edge-ordering heuristics. We ran the two approaches for edge ordering: *deg_sum* and *df_sum* (see Section 4) and report the degradation factor compared to the compression factor throughout the run of the algorithm. The results for the Cubicles scenario, normal setting with a drift value of 0.16 are summarized in Figure 7. One can see that the proposed heuristic, *df_sum*, obtained a lower average path degradation than the alternative, *deg_sum*, and that the error bars for *df_sum* are smaller. For additional heuristics that we investigated and more experimental results, the reader is referred to Shaharabani et al. (2013). All our experiments reported below refer to the *df_sum* edge ordering.

5.1.2. Evaluation of edge re-insertion. Recall that after an edge contraction was performed we proposed re-insertion of edges into the priority queue. These edges are the neighboring edges of the newly introduced edges. Namely, for each new edge $e = (u, p)$, where p is the contraction point of the contracted edge, we take all edges $e' = (u, v)$ that have already been removed from the queue, but could not be contracted and re-insert them into the queue. Clearly, this results in additional computation time; running the algorithm without edge re-insertion reduces the runtime by approximately 75%. To evaluate the effectiveness of edge re-insertion, we ran the algorithm on the Cubicles scenario in the normal setting with drift values of 0.8 and 0.16, both with and without edge re-insertion. The results are summarized in Figure 8, one can see that edge re-insertion allows for a much higher compression rate of the original graph by a factor of more than two for both drift values. The average path degradation is slightly higher but this seems negligible when compared to the additional compression obtained. The rest of the results presented here use edge re-insertion. As mentioned, edge re-insertion incurs substantial additional runtime on the algorithm. Figure 9 reports on the average runtime of RSEC on the four scenarios.

5.2. Connectivity, compression rate and path quality

5.2.1. Roadmap connectivity. We tested how the RSEC algorithm affects the connectivity of the roadmap. We sampled 1000 random free configurations (playing the role of start or goal) and tested for each, if it could be

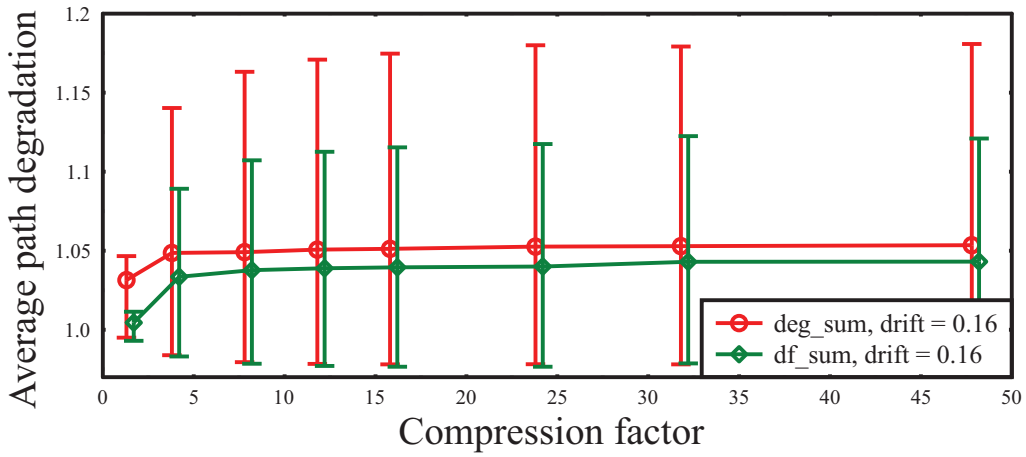


Fig. 7. Comparison of different heuristics on the Cubicles scenario, normal setting with a drift value of 0.16. Each point represents the average path degradation obtained when the algorithm reaches a certain compression factor. The error bars show the 20th and 80th percentile path degradation values. Note that the *y*-axis does not start at zero but is enlarged to contain only the range of the average path degradation that is plotted.

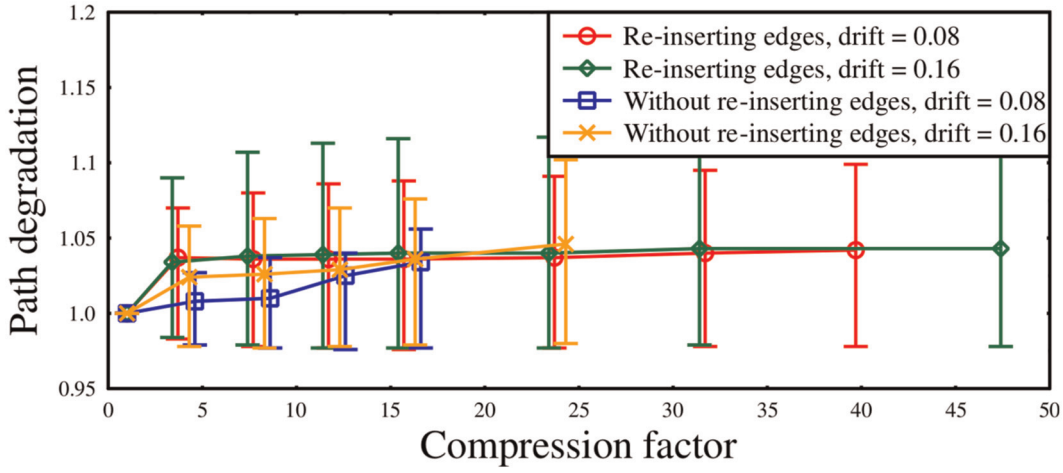


Fig. 8. Comparison of the edge re-insertion heuristic on the Cubicles scenario, normal setting with a drift values of 0.8 and 0.16. Each point represents the average path degradation obtained when the algorithm reaches a certain compression factor. The error bars show the 20th and 80th percentile path degradation values.

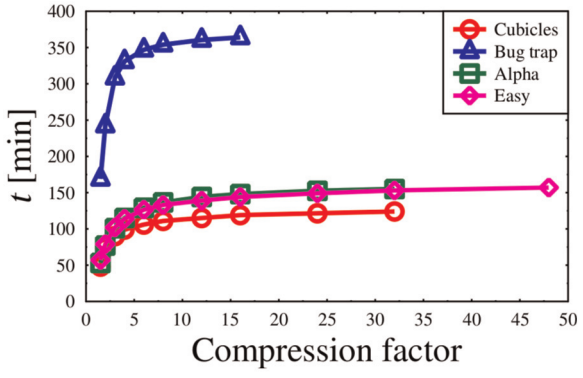


Fig. 9. Average runtime of RSEC with edge re-insertion as a function the compression factor obtained.

connected to the roadmap. Table 2 reports on the probability of connecting such a random query point to each roadmap. We compare the connectivity in the sparsified roadmap graph produced by RSEC with the original roadmap. Note that the connectivity is only dependent on the set of vertices, thus SPANNER (which does not remove vertices) has the same connectivity as the original roadmap. One can see that for all test cases, there is a negligible degradation in connectivity, if any.

5.2.2. Roadmap compression. The amount of compression achieved by each algorithm is governed by its input parameters—k for SPANNER, where k is a parameter related to the stretch, and drift bound for RSEC. Thus, for each algorithm we plot the average compression factor as a function

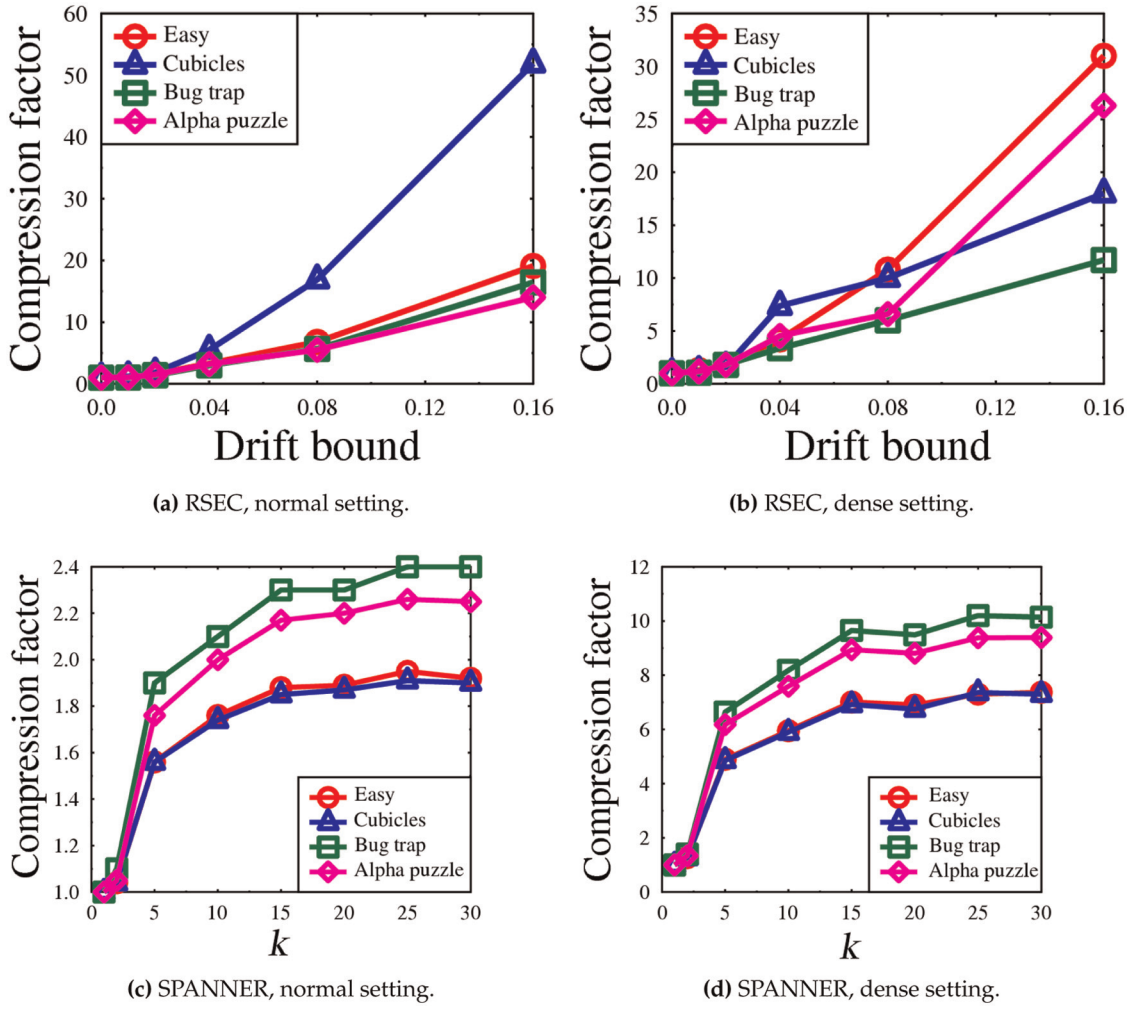
Table 2. Probability of connecting a random point to the roadmap. The drift bound used by RSEC is $d = 0.16$.

	Normal setting		Dense setting	
Scenario	Original	RSEC	Original	RSEC
Easy	100%	99.9%	100%	100%
Cubicles	98.9%	98.1%	100%	99.9%
Bug trap	100%	100%	100%	100%
Alpha puzzle	100%	100%	99.9%	99.9%

of its input parameter as shown in Figure 10. One can see that the compression factors achieved by RSEC are much higher than those achieved by SPANNER. Although we ran SPANNER with high values of its input parameter k, we did not manage to obtain a higher compression factor than 5 for the normal setting, and 10 for the dense setting. RSEC on the other hand exhibits very high compression factors, up to 48 for the cubicles scenario in the normal setting, that is, the sparsified graph is almost 2% of the size of the original roadmap.

One may ask what causes RSEC to terminate. Namely, why at some point in the course of the algorithm all examined edge contractions are illegal. For small drift values there is little freedom in moving vertices around, and maintaining the bounded drift invariant is what causes the algorithm to stop. As the drift bound grows, there is larger freedom in moving vertices around, which in turn increases the chances that the relocated edges attached to the moving vertices stop being collision-free. Consider for example the Cubicles scenario, in the normal setting: for the drift value of 0.2, 65% of illegal edge contractions are due to the drift bound, while this value reduces to 3% for a drift value of 0.16.

5.2.3. Roadmap quality. Obviously, the compression rate achieved by an algorithm should be examined together with the quality of the paths that can be extracted from the compressed roadmap graph. We examine the path quality as a function of the compression factor that was obtained by our algorithm. For example, Figure 11 shows a plot of these values for different drift values for the Cubicles scenario and Alpha puzzle scenario in the normal setting. One can see that the average value for each drift value is lower than 4% even for the high drift value. Moreover, the high



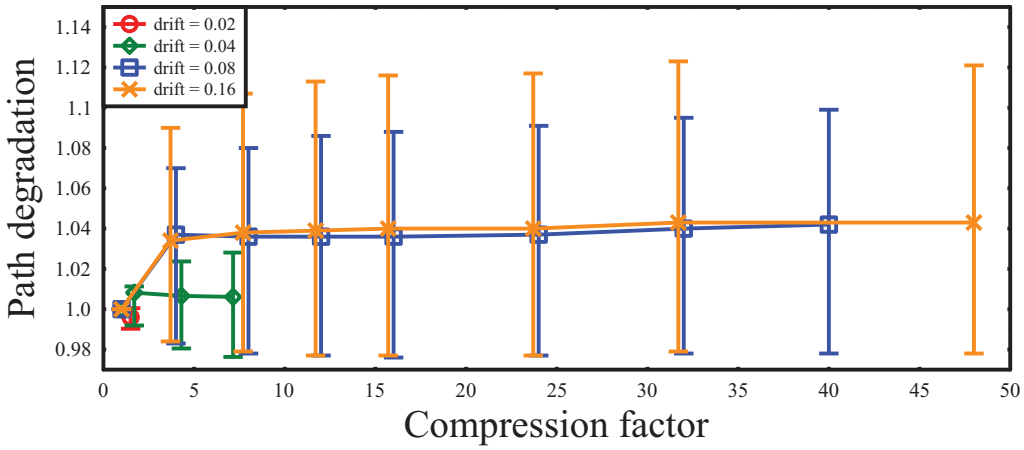
(a) RSEC, normal setting.

(b) RSEC, dense setting.

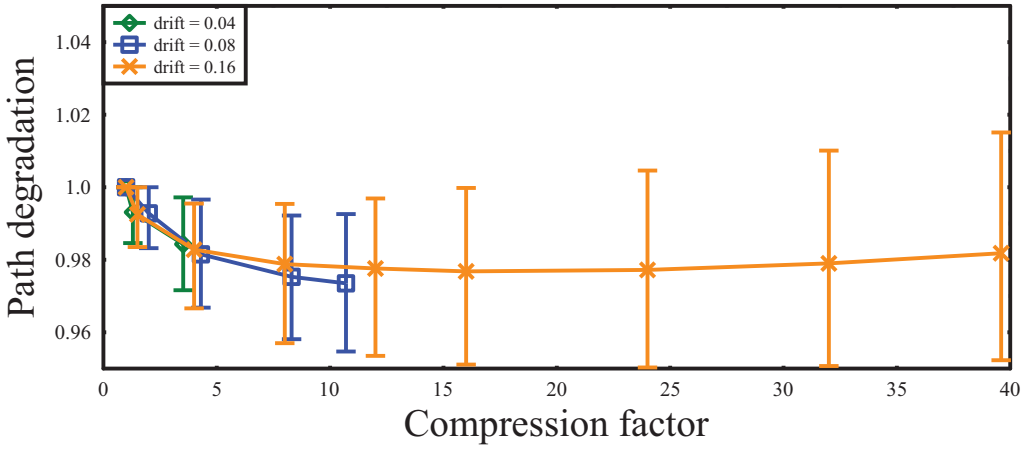
(c) SPANNER, normal setting.

(d) SPANNER, dense setting.

Fig. 10. Compression factor of the two sparsification algorithms for each scenario and each setting.



(a) Cubicles.



(b) Alpha puzzle.

Fig. 11. Path degradation as a function of the compression factor for RSEC, normal setting. Each point represents the average path degradation obtained when the algorithm reaches a certain compression factor. The error bars show the 20th and 80th percentile path degradation values.

Table 3. Maximal path degradation of RSEC for the normal setting.

Scenario	Drift = 0.02	Drift = 0.04	Drift = 0.08	Drift = 0.16
Easy	170%	243%	295%	397%
Cubicles	200%	300%	380%	570%
Bug trap	168%	223%	287%	350%
Alpha puzzle	180%	215%	225%	502%

error bars (denoting the path degradation for the 80th percentile value) are all lower than 11%. This indicates that the average path degradation is indeed a good indication of the path degradation obtained by the compressed graph. The same behavior is obtained for different scenarios and for the dense setting.

In order to consider the *worst-case* path degradation, we report in Table 3 on the maximal path degradation for each scenario. These are essentially outliers, as the value of the 80th percentile path degradation is significantly lower.

We also report on the roadmap quality obtained by the SPANNER algorithm. One can see in Figure 12 the results for the normal setting. For the same values of the compression factor, RSEC exhibits less degradation in average path quality. Having said that, the error bars are typically smaller in the SPANNER algorithm and we elaborate on this phenomenon next.

5.2.4. Path degradation with respect to path length. One can view a sequence of edge contractions as a process of

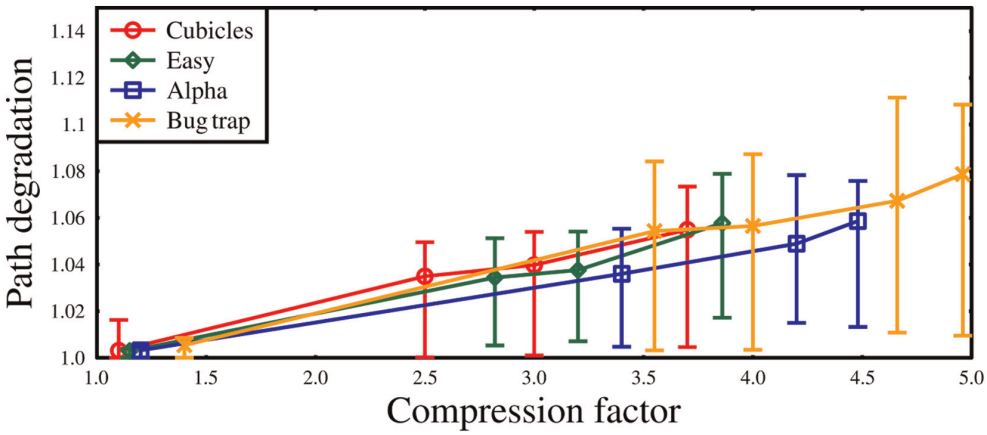
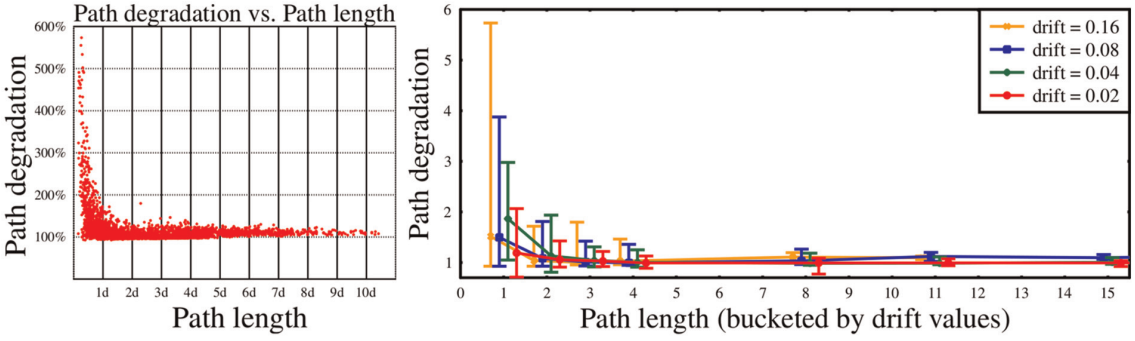


Fig. 12. Path degradation as a function of the compression factor for SPANNER, normal setting. Each point represents the average path degradation obtained when the algorithm reaches a certain compression factor. The error bars show the 20th and 80th percentile path degradation values.



(a) Plot for drift = 16%.

(b) Histogram for different drift values.

Fig. 13. Path degradation as a function of the path length for RSEC – Cubicles scenario, normal setting.

'smoothing' paths, a common post-processing practice in motion planning. This allows for the paths extracted by RSEC to be *shorter* than the original paths. For long paths this behavior is accentuated, namely paths become relatively even shorter, as more compression is applied. In contrast, for short paths, the degradation of paths typically increases as the compression grows. This seems natural as the effect of moving vertices is much more substantial on short paths.

To demonstrate this, we considered each of our paths as a 2D point. The x-coordinate of the point is the original path length and its y-coordinate is the path length after applying the RSEC algorithm. In Figure 13(a) we draw such a plot for the Cubicles scenario for a drift of 16%. We space the x-axis in intervals of drift-bound size. Below twice the drift bound we have high path degradation values, but these are outliers. For long paths, we can see that there are no outliers and the path degradation is negligible. To demonstrate this differently, we bucket the points and draw a histogram for different drift values. This is depicted in Figure 13(b) where the error bars indicate the maximal and minimal values in each bucket.

Considering the SPANNER algorithm, the behavior is somewhat the opposite. Short paths typically do not change at all, hence no degradation is seen. After a minimal path length is passed, the sparsification algorithm tends to remove edges and path degradation is observed. This is demonstrated in Figure 14.

5.2.5. Average degree. Recall that the average degree $\bar{k}$ of a graph $G = (V, E)$ is $\bar{k}(G) = \frac{2|E|}{|V|}$. As mentioned, each edge contraction on a graph $G_i = (V_i, E_i)$ reduces the number of vertices by one and the number of edges by at least one. Let $\kappa_i = |E_i| - |E_{i+1}|$, namely $\kappa_i - 1$ is the number of vertices adjacent to *both* endpoints of the edge contracted at the ith iteration. Thus, $\bar{k}(G_{i+1}) = \frac{|V_i|}{|V_i|-1} \cdot \bar{k}(G_i) - 2\frac{\kappa_i}{|V_i|-1}$. Thus the average degree will grow for $\kappa_i < \frac{\bar{k}(G_i)}{2}$ and shrink for $\kappa_i > \frac{\bar{k}(G_i)}{2}$.

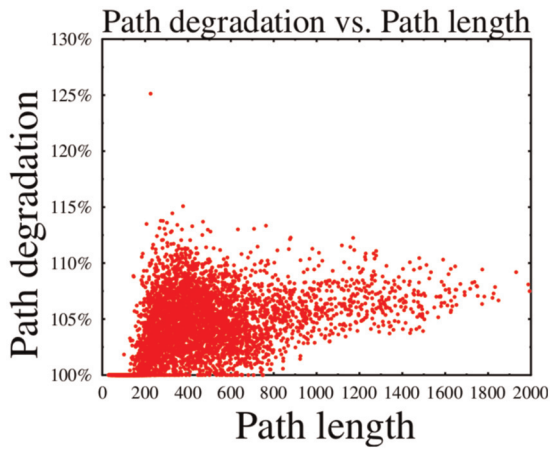


Fig. 14. Path degradation as a function of the path length for SPANNER – Cubicles scenario, normal setting. The plot is drawn for $k = 30$ which corresponds to a compression factor of 3.6.

Figure 15 depicts the average degree of the graph as a function of the compression factor. One can notice the interesting phenomenon (which we observed in other settings as well; results not shown) that the average degree increases for low compression values and then gradually decreases. This gives an indication that in the original graph, the number of edges shared between two adjacent nodes is small. Namely, for a fixed edge (u, v), the number of nodes r such that both edges (r, u) and (r, v) are in the roadmap is small. As RSEC performs a series of contractions, this value increases resulting in a decrease in the average degree.

5.2.6. The effect of multiple homotopy classes on coverage. We include a scene (Figure 16) that embodies multiple different homotopy classes and small visibility regions for many nodes. This tests if such scenarios may induce possible coverage problems for our algorithm. The tested coverage of the original roadmap was 100%, even for the highest compression ratios (the highest compression ratio reached was 32) the tested coverage of the compressed roadmap was above 99%.

5.2.7. The effect of narrow passages. A potential problem with having narrow passages in the free space is that edges at narrow passages (which may fail to contract due to obstacles) may be re-inserted many times, which may significantly increase the runtime. To test this hypothesis, we ran RSEC on two scenarios. Easy (Figure 4) and Twisty cool (also provided with the OMPL distribution). The two scenarios consist of an identical robot that needs to pass through a hole in a wall. The scenarios differ with respect to the hole size. The Twisty cool scenario is more complex because of the narrow passage induced by the hole.

Recall that an edge contraction can fail either due to violation of the bounded drift invariant or the fact that a newly introduced edge is not collision-free (we call this a *collision failure*). In both settings, the ratio of the number of collision failures and total failures is roughly 75%, suggesting that the structure of the C-space affects the behavior of the algorithm. The runtimes between the two scenarios differ slightly: the algorithm applied to the Easy scenario runs in 86% of the time it takes it to run on the Twisty cool scenario. However, when running RSEC on both scenarios without edge re-insertion, similar results were obtained—the algorithm applied to the Easy scenario runs in 93% of the time it runs on the Twisty cool scenario.

We believe that narrow passages do not pose too much difficulty for the RSEC algorithm because locally, within the narrow passage, an edge contraction is likely to succeed. Only at the 'entrance' to the narrow passage will an edge contraction fail with higher probability (due to possible changes in the local structure of the obstacles). Our experiments suggest that RSEC's performance might be affected by the *structure* of the C-space.

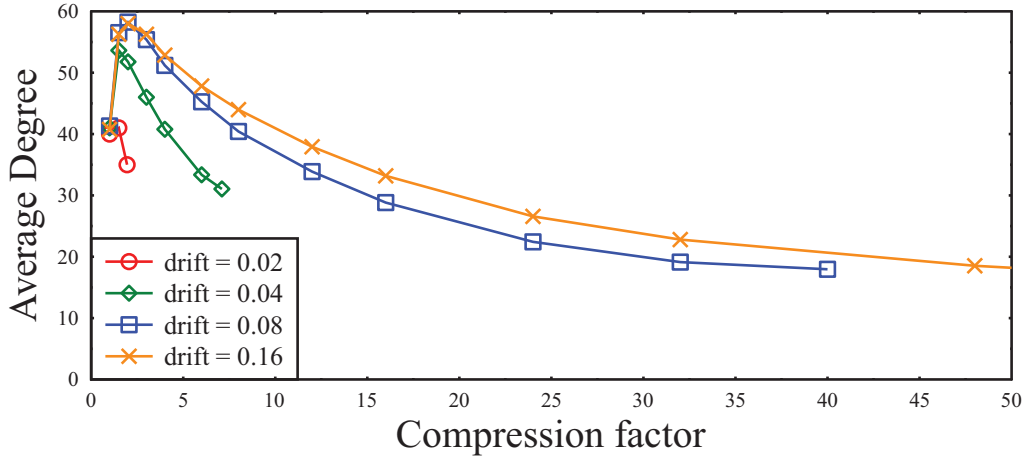


Fig. 15. Average degree for RSEC – Cubicles scenario, normal setting.

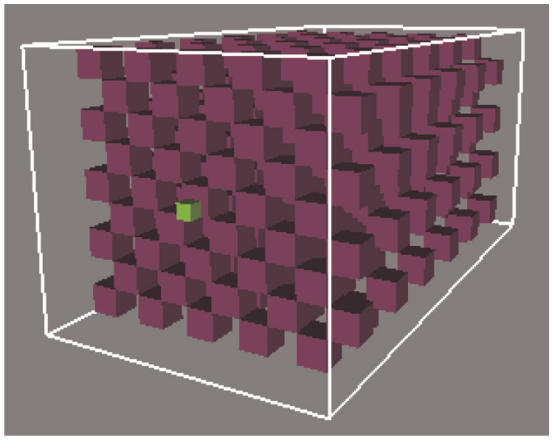


Fig. 16. Grids scenario for testing the effect of multiple homotopy classes on coverage. The square robot (green) is moving between a series of grid barriers (purple). The barriers induces multiple homotopy classes.

6. Conclusion

In this paper we described a simple, effective algorithm, based on edge contraction, to compress a roadmap. We focused on the length of a path as the measure of its quality. We believe that our bound on the degradation of paths can be adapted to quality measures such as *maximum bottleneck clearance* (maximizing the minimum distance between the path and obstacles) or *maximum inverse k-clearance*. (The inverse *k*-clearance of an edge is defined as the ratio of its length and the *k*th power of the minimum distance between the edge and the obstacles, and the inverse *k*-clearance of a path is the sum of the inverse *k*-clearances of the edges on the path (Raveh et al., 2011).)

There are several other directions for future research:

(i) Allow the contraction point of an edge to be located in the vicinity of the edge (and not on the edge itself). This will require computing the upper envelope of surfaces in 3D space for which the machinery is readily available (Fogel et al., 2012).

(ii) Allow an edge contraction even if not all neighbors can be connected to the contraction point — remove the edges that cannot be connected. If applied carefully, this may lead to higher compression rates with limited effect on path degradation.

(iii) The SPANNER algorithm performs better when the original graph is dense, so develop a hybrid algorithm by combining RSEC with SPANNER.

(iv) This paper considers the roadmap sparsification problem only in an offline setting. Extending the RSEC algorithm to an online setting requires a number of new ideas.

(v) Consider systems with complex dynamics and the roadmaps used by such systems.

Authors' note

A preliminary and abridged version of this paper appeared in the Proceedings of the 2013 IEEE International Conference on Robotics and Automation, pp. 4098–4105.

Acknowledgements

The authors thank Barak Raveh for his contribution at the early stages of the this work, and the anonymous reviewers for their useful comments.

Funding

Work by OS, DS and DH was supported in part by the 7th Framework Programme for Research of the European Commission (FET-Open grant number 255827, CGL—Computational Geometry Learning), by the Israel Science Foundation (grant number 1102/11), by the German–Israeli Foundation (grant number 1150-82.6/2011), and by the Hermann Minkowski–Minerva Center for Geometry at Tel-Aviv University. Work by PA was supported by the NSF (grant numbers CCF-09-40671, CCF-10-12254, and CCF-11-61359) by the US–Israel Binational Science Foundation (grant number 2012/229) by the ARO (contract number W911NF-13-P-0018) and by the ERDC (contract number W9132V-11-C-0003).

Notes

1. To simplify the exposition, we use the Euclidean distance to describe the algorithm. The algorithm can be generalized to the weighted combination of the Euclidean distance and the angular distance, as we demonstrate in Section 5.
2. In our context, a local planner is a predicate that determines whether there exists a collision-free path between two configurations.

References

Agarwal PK, Har-Peled S and Varadarajan KR (2005) Geometric approximation via coresets. In: Goodman JE, Pach J and Welzl E (eds) *Combinatorial and Computational Geometry*. Cambridge: Cambridge University Press, pp. 1–30.

Agarwal PK, Sharathkumar R and Yu H (2009) Approximate Euclidean shortest paths amid convex obstacles. In: *20th ACM-SIAM symposium on discrete algorithms (SODA '09)*, New York, USA, 4–6 January, pp. 283–292. New York: ACM Press.

Agarwal PK and Suri S (1998) Surface approximation and geometric partitions. *SIAM Journal on Computing* 27(4): 1016–1035.

Aleksandrov L, Maheshwari A and Sack JR (2005) Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM* 52(1): 25–53.

Arya S, Das G, Mount DM, Salowe JS and Smid M (1995) Euclidean spanners: Short, thin, and lanky. In: *27th annual ACM symposium on the theory of computing* (STOC 95), Las Vegas, Nevada, USA, 29 May-1 June, pp. 489–498. New York: ACM Press.

Choset H, Lynch KM, Hutchinson S, Kantor G, Burgard W, Kavraki LE, et al. (2005) *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge: MIT Press.

Clarkson KL (1987) Approximation algorithms for shortest path motion planning. In: *21st annual ACM symposium on the theory of computing* (STOC' 87), New York, New York, USA, pp. 56–65. New York: ACM Press.

Cormode G, Garofalakis M, Haas PJ and Jermaine C (2012) Synopses for massive data: Samples, histograms, wavelets and sketches. *Foundations and Trends in Databases* 4(1–3): 1–294.

Dobson A and Bekris KE (2013) Improving sparse roadmap spanners. In: *IEEE international conference on robotics and automation (ICRA '13)*, Karlsruhe, Germany, 6-10 May, pp. 4106–4111. Piscataway: IEEE Press.

Dobson A, Krontiris A and Bekris KE (2012) Sparse roadmap spanners. In: *10th workshop on the algorithmic foundations of robotics* (WAFR' 12), Cambridge, MA, USA, 13-15 June, pp. 279–296.

Edelsbrunner H (2001) *Geometry and Topology for Mesh Generation*. Cambridge: Cambridge University Press.

Elkin M and Solomon S (2013) Optimal Euclidean spanners: Really short, thin and lanky. In: *45th annual ACM symposium on the theory of computing* (STOC' 13), Palo Alto, CA, USA, 1-4 June, pp. 645–654. New York: ACM Press.

Fogel E, Halperin D and Wein R (2012) *CGAL Arrangements and their Applications – A Step-by-Step Guide*. Heidelberg: Springer.

Garland M and Heckbert PS (1997) Surface simplification using quadric error metrics. In: *24th annual conference on computer graphics and interactive techniques (SIGGRAPH '97)* Los Angeles, CA, USA, 3-8 August, pp. 209–216. New York: ACM Press.

Geraerts R and Overmars MH (2007) Creating high-quality paths for motion planning. *International Journal of Robotics Research* 26(8): 845–863.

Har-Peled S (1999) Constructing approximate shortest path maps in three dimensions. *SIAM Journal on Computing* 28(4): 1182–1197.

Hoppe H, DeRose T, Duchamp T, McDonald J and Stuetzle W (1993) Mesh optimization. In: *20th annual conference on computer graphics and interactive techniques (SIGGRAPH '93)* Anaheim, CA, USA, 2-6 August, pp. 19–26. New York: ACM Press.

Hsu D, Latombe JC and Motwani R (1999) Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications* 9(4–5): 495–512.

Jaillet L and Siméon T (2006) Path deformation roadmaps. In: *7th international workshop on the algorithmic foundations of robotics (WAFR 2006)*, New York, USA, 16–18 July, pp. 19–34. Heidelberg: Springer.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research* 30(7): 846–894.

Kavraki LE, Svestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics* 12(4): 566–580.

Kuffner JJ and Lavalle SM (2000) RRT-Connect: An efficient approach to single-query path planning. In: *IEEE international conference on robotics and automation (ICRA '00)*, pp. 995–1001. Piscataway: IEEE Press.

Lien JM, Thomas SL and Amato NM (2003) A general framework for sampling on the medial axis of the free space. In: *IEEE international conference on robotics and automation (ICRA '03)* San Francisco, CA, USA, 24-28 April, pp. 4439–4444. Piscataway: IEEE Press.

Luebke D, Watson B, Cohen JD, Reddy M and Varshney A (2002) *Level of Detail for 3D Graphics*. New York: Elsevier.

Marble JD and Bekris KE (2011) Computing spanners of asymptotically optimal probabilistic roadmaps. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS '11)*, San Francisco, CA, 25-30 September, pp. 4292–4298. Piscataway: IEEE Press.

Marble JD and Bekris KE (2012) Towards small asymptotically near-optimal roadmaps. In: *IEEE international conference on robotics and automation (ICRA '12)*, St. Paul, MN, USA, 14-18 May, pp. 2557–2562. Piscataway: IEEE Press.

Marble JD and Bekris KE (2013) Asymptotically near-optimal planning with probabilistic roadmap spanners. *IEEE Transactions on Robotics* 29(2): 432–444.

Narasimhan G and Smid M (2007) *Geometric Spanner Networks*. Cambridge: Cambridge University Press.

Nechushtan O, Raveh B and Halperin D (2010) Sampling-diagram automata: A tool for analyzing path quality in tree planners. In: *9th workshop on the algorithmic foundations of robotics* (WAFR' 2010), Singapore, 13-15 December, pp. 285–301.

Nieuwenhuisen D and Overmars MH (2004) Useful cycles in probabilistic roadmap graphs. In: *IEEE international conference on robotics and automation (ICRA '04)*, Barcelona, Spain, 18-22 April, pp. 446–452. Piscataway: IEEE Press.

Peleg D and Schäffer AA (1989) Graph spanners. *Journal of Graph Theory* 13(1): 99–116.

Raveh B, Enosh A and Halperin D (2011) A little more, a lot better: Improving path quality by a path-merging algorithm. *IEEE Transactions on Robotics* 27(2): 365–371.

Schmitzberger E, Bouchet JL, Dufaut M, Wolf D and Husson R (2002) Capture of homotopy classes with probabilistic road map. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS '02)*, EPFL, Switzerland, 30 September-4 October, pp. 2317–2322. Piscataway: IEEE Press.

Shaharabani D, Salzman O, Agarwal PK and Halperin D (2013) Sparsification of motion-planning roadmaps by edge contraction. In: *IEEE international conference on robotics and automation (ICRA '13)*, Karlsruhe, Germany, 6-10 May, pp. 4098–4105. Piscataway: IEEE Press.

Sharir M and Agarwal PK (1995) *Davenport–Schinzel Sequences and their Geometric Applications*. Cambridge: Cambridge University Press.

Siméon T, Laumond J and Nissoux C (2000) Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics* 14(6): 477–493.

Şucan IA, Moll M and Kavraki LE (2012) The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4): 72–82.

Varadarajan KR and Agarwal PK (2000) Approximating shortest paths on a nonconvex polyhedron. *SIAM Journal on Computing* 30(4): 1321–1340.

Wang W, Balkcom DJ and Chakrabarti A (2013) A fast streaming spanner algorithm for incrementally constructing sparse roadmaps. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS '13)*, Tokyo, Japan, 3-8 November, pp. 1257–1263. Piscataway: IEEE Press.