Project number IST-25582

CGL

Computational Geometric Learning

Sparsification of Motion-Planning Roadmaps by Edge Contraction

STREP

Information Society Technologies

Period covered:	November 1, 2011–October 31, 2012
Date of preparation:	October 21, 2012
Date of revision:	October 21, 2012
Start date of project:	November 1, 2010
Duration:	3 years
Project coordinator name:	Joachim Giesen (FSU)
Project coordinator organisation:	Friedrich-Schiller-Universität Jena Jena, Germany

Sparsification of Motion-Planning Roadmaps by Edge Contraction

Doron Shaharabani*, Oren Salzman*, Pankaj K. Agarwal† and Dan Halperin*
* Balvatnic School of Computer Science, Tel-Aviv University, Israel
† Department of Computer Science, Duke University, USA

***Abstract*—We present Roadmap Sparsification by Edge Contraction (RSEC), a simple and effective algorithm for reducing the size of a motion-planning roadmap. The algorithm exhibits minimal effect on the quality of paths that can be extracted from the new roadmap. The primitive operation used by RSEC is *edge contraction*—the contraction of a roadmap edge to a single vertex and the connection of the new vertex to the neighboring vertices of the contracted edge. For certain scenarios, we compress more than 98% of the edges and vertices at the cost of degradation of average shortest path length by at most 2%.**

I. Introduction

The introduction of sampling-based planners [1], [2], [3] enabled solving motion-planning problems that were previously infeasible [4]. Specifically, for multi-query scenarios, planners such as the Probabilistic Roadmap Planner (PRM) [2] approximate the connectivity of the free space ($\mathcal{C}_{\text{free}}$) by taking random samples from the Configuration Space (C-space) and connecting near-by free configurations when possible. The resulting data structure, the *roadmap*, is a graph where 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 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 [5], [6] additional work is required.

Smoothing is a common practice that may improve the quality of a single path. In particular when applied to a path extracted from the output roadmap of an algorithm that produces paths deformable to optimal ones (see, e.g., [7], [8]). However smoothing is costly and needs to be applied for every query. Nieuwenhuisen and Overmars [9] suggested adding useful cycles to the roadmap in order to improve the resulting quality of the paths. Raveh et al. [10] proposed combining the output of several different planners. Recent work by Karaman and Frazzoli [6] introduced several sampling based algorithms, including PRM*, such that asymptotically, the solution returned by the roadmap almost surely converges to an optimum path (with regards to some quality measure). This desired behavior comes at the expense of increasing the number of neighbors each node is connected to, as the number of samples increases.

Thus, motion-planning algorithms that create high-quality roadmaps result in large, dense graphs. This may be undesirable due to prohibitive storage requirements and long online query processing time. We seek a compact representation of the roadmap graph without sacrificing the desirable guarantees on path quality.

A. Related work

Geraerts and Overmars [11] suggested an algorithm for creating small roadmaps of high-quality but their technique is limited to two- and three-dimensional C-spaces. The work on graph spanners [12], [13] is closely related to our problem of computing a compact representation of a roadmap graph. Formally, a t-spanner of a graph $\mathcal{G} = (V, E)$ is a sparse subgraph $\mathcal{G}' = (V, E' \subseteq E)$, where the shortest path between any pair of points in $\mathcal{G}'$ is no longer than t times the shortest path between them in $\mathcal{G}$. The parameter t is referred to as the *stretch* of the spanner. It is well known that small size $(1+\epsilon)$-spanners exist for complete Euclidean graphs [13], [14], [15]. 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 [16], [17], [18], [19]. In the context of roadmap constructions, Marble and Bekris [20] 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. Recently, graph-spanner algorithms have also been incorporated in the construction phase of the roadmap itself [21], [22], [23].

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 will be useful to remove redundant vertices as well and to construct a small-

Work by D.S., O.S., and D.H. has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the Israel Science Foundation (grant no. 1102/11), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

Work by P.A. was supported by NSF under grants IIS-07-13498, CCF-09-40671, CCF-10-12254, and CCF-11-61359,by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, and by an ARL award W9132V-11-C-0003.

size graph in the free space. Such an approach was recently proposed for computing approximate shortest paths for a point robot amid convex obstacles in two or three dimensions [24], 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. 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. The surface-simplification problems asks for simplifying the mesh—reducing its size—while ensuring that the resulting surface approximates the original one within a prescribed error tolerance [25]. Some versions of the optimal surface-simplification problem, the problem of computing a smallest-size surface, are known to be NP-Hard [26], and several approximation algorithms and heuristics have been proposed. 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 by now become the most popular method for simplifying a surface [27], [28], [29].

B. Contribution

In this work, we adapt the widely used *edge-contraction* technique for surface simplification to roadmap simplification. We suggest a simple algorithm to sparsify a roadmap—Roadmap Sparsification by Edge Contraction (RSEC) where edge contraction is the primitive operation. Our algorithm often exhibits little degradation in path quality when compared to the original graph while providing very sparse graphs. In contrast to the algorithm of Marble and Bekris [20] in which the set of vertices remains intact, RSEC dramatically reduces the number of vertices.

In Section II we present the algorithmic framework and in Section III we cover the implementation details. Section IV presents experimental results comparing our implementation with several alternatives. For certain scenarios, we compress more than 98% of the edges and vertices while causing degradation of average path length by at most 2%. Additionally, we compare RSEC with the algorithm presented by Marble and Bekris [20]. 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 with a discussion and suggestions for further research in Section V.

II. Algorithmic framework

Let $\mathcal{G} = (V, E)$ be an undirected input graph, which is the output of a PRM-type algorithm approximating $\mathcal{C}_{\text{free}}$. Informally, we wish to construct a graph $\mathcal{G}' = (V', E')$ such that $\mathcal{G}'$ is a more 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 work we concentrate on the quality measure of path length.

Algorithm 1 is_contractible $(\mathcal{G}(E,V),(u,v),p)$

```
1: for all w ∈ {G.neighbors(v) ∪ G.neighbors(u)} do
2:     if local_planner(w, p) = FAILURE then
3:         return FAILURE
4: return SUCCESS
```

Algorithm 2 edge_contraction $(\mathcal{G}(E,V),(u,v),p)$

```
1: if is_contractible(G, (u, v), p) then
2:     for all v' ∈ {G.neighbors(v)} do
3:         E ← E ∪ {(v', p)}
4:         E ← E \ {(v', v)}
5:     for all u' ∈ {G.neighbors(u)} do
6:         E ← E ∪ {(u', p)}
7:         E ← E \ {(u', u)}
8:     V ← V ∪ {p}
9:     V ← V \ {v, u}
10:    return SUCCESS
11: else
12:    return FAILURE
```

We denote the set of all neighbors of a vertex $v \in V$ by:

$$\text{neighbors}(v) = \{u \in V | (u,v) \in E\}.$$

Given an edge (u,v) and a point $p \in \mathcal{C}_{\text{free}}$, we define an *edge contraction* of (u,v) to p as the following process: Adding p as a new vertex, adding an edge (w,p) for each vertex $w \in \{\text{neighbors}(u) \cup \text{neighbors}(v)\}$ and removing the vertices u, v and all their neighboring edges from the graph. We say that u and v were contracted to p and define:

$$\text{parent}(v') = \{v | v \text{ was contracted to } v'\}.$$

Using this definition, we can recursively define that a vertex v is an *ancestor* of a vertex v' if either: (i) v is a parent of v' or (ii) there exists a vertex u such that u is the parent of v' and v is an ancestor of u.

An edge contraction is considered *legal* if each new edge is collision-free. This is detailed in Algorithm 1, which performs the validity check that an edge (u,v) can be contracted to a point p using a local planner[1] and in Algorithm 2, which updates the graph $\mathcal{G}$ after the contraction.

Connecting an edge (u,v) to a graph $\mathcal{G}$ reduces the size of $\mathcal{G}$—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).

Our algorithm, Roadmap Sparsification by Edge Contraction (RSEC), performs a series of legal edge contractions as detailed in Algorithm 3. The algorithm maintains an order on the edges considered for contraction by using a priority queue ordered according to some weight function (line 2).

[1] A local planner is a predicate that determines if there exists a collision-free path between two configurations.

Algorithm 3 RSEC ($\mathcal{G}(E,V)$)

1: $\mathcal{G}' \leftarrow \mathcal{G}$
2: $Q \leftarrow$ initialize_queue(E)
3: **while** not_empty(Q) **do**
4: $e \leftarrow Q$.pop_head
5: $p \leftarrow$ get_contraction_point(e)
6: **if** $p \neq$NIL **then**
7: **if** edge_contraction ($\mathcal{G}', e, p$) == SUCCESS **then**
8: $Q \leftarrow$ update_queue(E')

Algorithm 4 get_contraction_point ($e = (u, v)$)

1: $p \leftarrow$ random_point(u, v)
2: **if** collision_detector$(p) \neq$ FREE **then**
3: **return** NIL
4: **for all** $w \in u$.ancestors() $\bigcup$ v.ancestors() **do**
5: **if** distance$(w, p) > d$ **then**
6: **return** NIL
7: p.ancestors() $\leftarrow u$.ancestors() $\bigcup$ v.ancestors()
8: **return** p

At each step the current edge is popped out of the queue (line 4) and a contraction point is computed (line 5). If the prospective contraction point is valid and the edge contraction was successful (lines 6, 7), the queue is updated, as the weights of edges may have changed (line 8). The process terminates when the queue is empty.

III. Algorithmic details

Algorithm 3 presents RSEC as a general, modular framework. As such, some technical details still need to be addressed in order to complete its description. Specifically, what point should an edge be contracted to? How do we order the edges in our queue to obtain a sparse graph of high quality?

A. Contraction point choice

When contracting an edge e to a point p, several alternatives regarding the choice of p come to mind: (i) The midpoint of e, (ii) an incident vertex of e or (iii) a random point along e. One may also consider several contraction points when testing whether an edge is contractible.

The advantage of alternative (ii) is that the neighboring edges of the vertex chosen to be the contraction point are not altered. This reduces dramatically the number of collision checks needed by the algorithm. Alternatives (i) and (iii), on the other hand, seem to distribute naturally the location of the contraction point as the average of its ancestors. We chose the contraction point to be alternative (iii), random point along the edge, as we observed higher path degradation in comparison to the other alternatives.

B. Quality-driven constraint

As we wish to preserve the quality and connectivity of the output graph we add the constraint that vertices in the new graph are not too far from their ancestors. Then, given a *drift bound* $d > 0$ and a distance function $dist$, we maintain the following invariant in our algorithm:

> **Bounded drift invariant** - For every vertex $v' \in V'$ and for every $v \in$ ancestor(v'), $dist(v', v) \leq d$.

Line 5 of Algorithm 3 performs a call to a subroutine get_contraction_point(e). The subroutine verifies that the contraction point does not violate the bounded drift invariant. Algorithm 4 demonstrates an implementation of this subroutine.

We note that in practice we normalize the drift bound to be $\frac{d}{a}$ where a is the length of the workspace bounding box diagonal. In the rest of the paper we use the term drift bound to refer to the normalized drift bound.

C. Edge ordering

We suggest to choose the edges next edge $e = (u, v)$ for contraction as follows: We order (from low to high) the edges according to the sum of degrees of the vertices incident to the edge, namely degree(u) + degree(v), and name the heuristic *deg_sum*. The motivation for this heuristics comes from the fact that vertices with low degree impose less validity checks (and hence less constraints) when checking whether an edge is contactable. This not only reduces the computation time, but more importantly, increases the probability of a valid contraction.

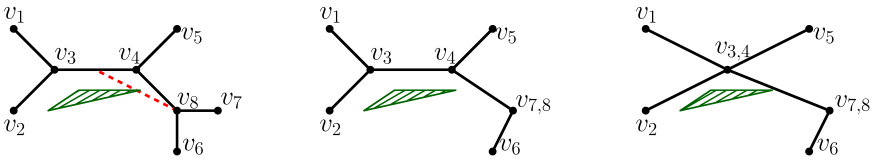
In Section IV we evaluate our heuristic by suggesting several alternatives and comparing them to deg_sum. Indeed, in the experimental results deg_sum outperforms the alternative heuristics.

An additional implementation issue that should be addressed regards the question "should edges be re-inserted into the priority queue?" An edge (u, v) that was removed from the queue because it could not be contracted could possibly be contracted at a later stage of the algorithm. Such an event could occur when a vertex that is adjacent to either u or v moves as a result of a contraction operation on a different edge. An example of such a scenario is depicted in Figure 1. Obviously, re-inserting these edges is costly with regards to runtime but improves the compression achieved. As we did not consider processing time to be a major constraint, we allowed edges to be re-inserted to the queue.

IV. Evaluation

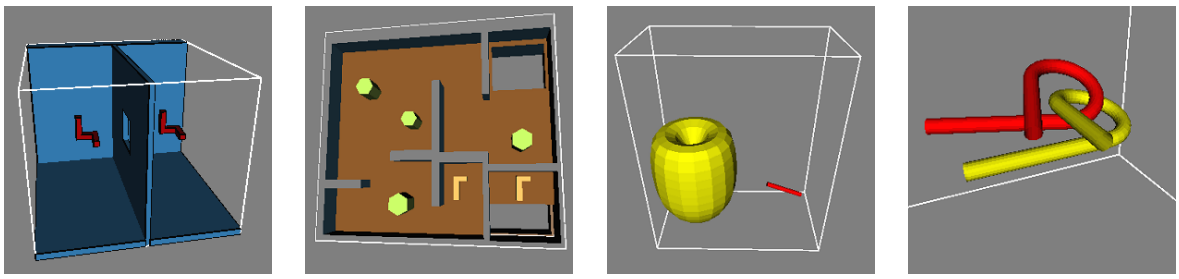
We evaluated our algorithm by running the PRM* [6] algorithm on several scenarios to obtain initial roadmaps. We first compared our heuristic for ordering the edges with several alternatives. Then we compared RSEC with the spanner-based sparcification algorithm by Marble and Bekris [20] (which we will call SPANNER). We chose to compare our algorithm to SPANNER as it is, to the best of our knowledge, the only offline algorithm for reducing roadmaps.

We measure the quality of a roadmap in terms of path length and report two values: (i) *path degradation* and (ii) *compression factor*. The former is the ratio of the average shortest



(a) (b) (c)

Fig. 1. 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 $v_{3,4}$.



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

Fig. 2. Environments used for the experiments. The robot is depicted in several configurations in each environment. All scenarios were taken from the OMPL [30] distribution.

path length in the sparse graph and the average shortest path length in the original graph for random queries. The latter is the ratio $\frac{|\mathcal{G}|}{|\mathcal{G}'|}$ where $|\mathcal{G}|$ be the size[2] of the original roadmap and $|\mathcal{G}'|$ be the size of the compressed roadmap.

All experiments were run using the Open Motion Planning Library (OMPL) [30] on a 3.4GHz Intel Core i7 processor with 8GB of memory. We used several scenarios provided by the OMPL distribution. The scenarios are depicted in Figure 2.

A. Heuristic evaluation

In Section III we described our edge ordering heuristic, namely deg_sum. Several natural alternatives may come to mind: The first, termed *compressibility*, chooses the next edge $e = (u, v)$ to contract according to how many edges could potentially be eliminated from the graph by contracting e into a single point. The number of edges that will be eliminated (except from the contracted edge) is the number of common neighbors of u and v. We normalize this value by dividing by the total number of neighbors of u and v and define the compreassbility of (u, v) as:

$$\text{compressibility}(u, v) = \frac{|\text{neighbors}(u) \cap \text{neighbors}(v)|}{|\text{neighbors}(u) \cup \text{neighbors}(v)|}.$$

The second heuristic that one may think of chooses the next edge to contract according to its *clearance*, namely the distance of the edge from the closest obstacle (ordered from high clearance to low). This is motivated by the fact that edges of high-clearance may be good candidates for successful contractions. A third option may be a simple first in first out (FIFO) ordering scheme.

We ran the alternative implementations on roadmaps containing 5,000 vertices and approximately 50,000 edges. The results for the Cubicles scenario are summarized in Figure 3.

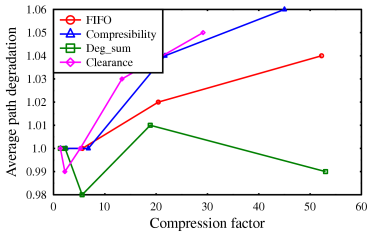


Fig. 3. Comparison of different heuristics on the cubicles scenario.

For low compression factors, all heuristics exhibit no degradation with regards to path quality. A possible explanation is that for small perturbations, our algorithm may be seen as a smoothing of the graph. For high compression factors, the path degradation is very low for all heuristics, where deg_sum seems to be the best choice.

One can see that deg_sum and FIFO achieve the highest compression factors. In order to explain this, we note that vertices with high degrees are more likely to fail a contraction operation, thus providing a negative influence to the overall compression achieved by RSEC. The clearance heuristic prioritizes edges with high clearance, resulting in contraction operations being applied repeatedly to high-clearance areas of the roadmap. As each contraction operation creates a vertex with a higher degree than its ancestors', biasing the contraction operations to specific areas of the graph at a time

[2]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. Thus for our scenarios in SE03 we measure the size of a roadmap $\mathcal{G}(V, E)$ as $|\mathcal{G}| = 6|V| + 3|E|$.

Scenario	Normal setting		Dense setting	
	PRM*	RSEC	PRM*	RSEC
Easy	100%	100%	100%	99.9%
Cubicles	99.9%	97.2%	100%	97.6%
Bug trap	100%	100%	100%	100%
Alpha Puzzle	100%	100%	100%	99.9%

TABLE II
Probability to connect a random point to the roadmap. The drift bound used by RSEC is $d = 0.16$.

is likely to create high degree vertices. In a similar manner, the compressibility is likely to focus the contractions on highly connected areas. On the other hand, the FIFO heuristic performs the contraction operations without prioritizing any specific areas of the graph, and the deg_sum heuristic aims to prevent high-degree vertices by concentrating on edges who's incident vertices have a low degree

Table I summarizes the algorithm's results on all scenarios for the deg_sum heuristic. For the complete set of results, we refer the reader to the appendix.

B. Comparison with SPANNER

When comparing RSEC with SPANNER, we used the same scenarios depicted in Figure 2 with two initial roadmap sizes. The first, which we call *normal setting*, contains a roadmap with 5,000 vertices and approximately 50,000 edges. The second, which we call *dense setting*, contains 20,000 vertices and approximately 1.2 million edges. The dense setting resembles the benchmark used by Marble and Bekris [20]. The crucial difference between our two settings is not the size but the average degree of each vertex (20 and 120 for the normal setting and dense setting, respectively).

Roadmap Connectivity We desire that a sparsification algorithm retains some quality measures while not sacrificing the ability to connect queries to the roadmap. Hence, we sampled 1000 random free configurations (playing the role of start or goal) and tested for each if it can be connected to the roadmap. Table II reports on the probability to connect such a random query point to each roadmap. As SPANNER does not remove vertices, the probability to connect a random point to the new roadmap does not change. Hence, we only present comparison results of RSEC and the original roadmap. One can see that for all test cases, there is a negligible degradation in connectivity (if any).

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 compression factor as a function of its input parameter as shown in Figure 4. One can see that the compression factors achieved by RSEC are much higher than SPANNER. Although we ran SPANNER with high values of its input parameter k, we did not manage to obtain a higher compression factor than 2.4 for the normal setting and 10 for the dense setting. RSEC on the other hand exhibits very high compression factors, up to 55 for the cubicles scenario in the normal setting (this is a sparsified graph which is less than 2% the size of the original roadmap).

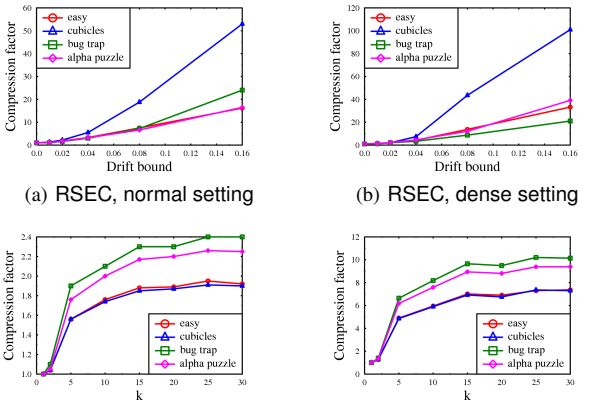
Roadmap Quality As the compression factor of the two algorithms is based on different parameters, we chose to compare the path quality as a function of the compression factor that was obtained. Figure 5 plots the average degradation in path length as a function of the compression factor for each algorithm for the normal setting. For the same values of the compression factor, RSEC exhibits less degradation in average path quality. For example in the bugtrap scenario, for the compression factor of 2.4, SPANNER exhibits a degradation in average path quality of around 5% while RSEC slightly improves the average quality of paths. The same behavior is observed for the dense setting depicted in Figure 6.

V. Discussion

Comparing the difference between RSEC and SPANNER helps understand both the advantages as well as the shortcomings of RSEC. A key drawback of SPANNER is that it overlooks the fact that the graph on which it operates is a roadmap. Thus, it does not introduce any new vertices and edges and is only capable of removing edges while leaving the set of vertices intact. In contrast, RSEC makes use of the fact that the roadmap is an approximation of some space and uses both a *collision detector* and a *local planner* to perform edge contraction. Clearly these operations are much more time-consuming than edge removal operations, but in return these natural operations in motion-planning algorithms grant RSEC much more power as demonstrated in Section IV.

The additional running time incurred by the geometric operations seems reasonable as RSEC is expected to run in an offline phase where running time is not the primary concern.

We saw that the performance of RSEC clearly surpasses SPANNER for graphs with a small average degree. This seems natural as the notion of edge contraction is borrowed from mesh simplifications where the average degree is at most 6. As the roadmaps become dense the advantage of RSEC diminishes. A possible approach may be to run the SPANNER



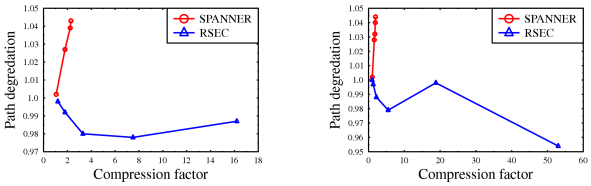
(a) RSEC, normal setting (b) RSEC, dense setting

(c) SPANNER, normal setting (d) SPANNER, dense setting

Fig. 4. Compression factor of the two sparsification algorithms for each scenario and each setting. Results are averaged over 5 runs.

Scenario		drift bound original	$d = 0.1$	$d = 0.2$	$d = 0.4$	$d = 0.8$	$d = 0.16$
Easy	average path length	314.8	99.8%	99.2%	98.1%	97.5%	99.1%
	compression factor	1	1.16	1.82	3.37	11.4	17.6
	vertices	5016	87.2%	59.0%	26.2%	10.1%	3.6%
	edges	48866	85.7%	54.0%	30.4%	14.0%	6.1%
Cubicles	average path length	472.4	99.9%	99.5%	98.1%	101.2%	99.2%
	compression factor	1	1.33	2.27	5.58	18.85	53
	vertices	5004	76.8%	45.6%	16.4%	5.3%	2.0%
	edges	45217	74.8%	43.7%	18.3%	5.3%	1.9%
Bug trap	average path length	41.8	100%	99.6%	97.6%	95.1%	93.8%
	compression factor	1	1.03	1.6	2.9	7.2	25.8
	vertices	5011	95.8%	66.9%	33.3%	12.3%	3.4%
	edges	66053	97.1%	61.6%	33.4%	14.1%	3.9%
Alpha puzzle	average path length	127.8	99.9%	99.3%	98.1%	97.1%	97.6%
	compression factor	1	1.09	1.74	3.23	6.57	16.3
	vertices	5053	92.1%	61.3%	28.3%	11.6%	4.1%
	edges	62294	91.8%	56.8%	31.4%	15.8%	6.5%

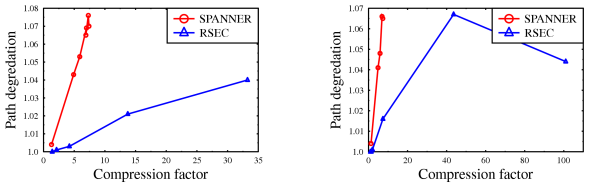
TABLE I
Average path degradation and compression factor of RSEC with respect to the original graph using the deg_sum heuristic. The results are averaged over 5 runs and 150 random queries.



(a) Easy (b) Cubicles

(c) Bug trap (d) Alpha puzzle

Fig. 5. Path degradation for each scenario as a function of the compression factor for each algorithm – Normal setting.



(a) Easy (b) Cubicles

(c) Bug trap (d) Alpha puzzle

Fig. 6. Path degradation for each scenario as a function of the compression factor for each algorithm — Dense setting.

algorithm with a small stretch factor in order to reduce the average degree (this is bound to happen as the number of vertices does not change) and then run RSEC.

Additional suggestions for further research include (i) deriving theoretical bounds to the quality of the obtained roadmap and (ii) relaxing the edge contraction operation to allow edge contractions even when not all neighbors can be connected to the contraction point. This may lead to higher compressions with limited effect on the path degradation.

REFERENCES

[1] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *IJCGA*, vol. 9, no. 4/5, 1999.

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

[3] J. J. Kuffner and S. M. Lavalle, "RRT-Connect: An efficient approach to single-query path planning," in *ICRA*, 2000, pp. 995–1001.

[4] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. MIT Press, June 2005.

[5] O. Nechushtan, B. Raveh, and D. Halperin, "Sampling-diagram automata: A tool for analyzing path quality in tree planners," in *WAFR*, 2010, pp. 285–301.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, vol. 30, no. 7, pp. 846–894, 2011.

[7] E. Schmitzberger, J.-L. Bouchet, M. Dufaut, W. Didier, and R. Husson, "Capture of homotopy classes with probabilistic road map," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[8] L. Jaillet and T. Siméon, "Path deformation roadmaps," in *WAFR*, 2006.

[9] D. Nieuwenhuisen and M. H. Overmars, "Useful cycles in probabilistic roadmap graphs," in *ICRA*, 2004, pp. 446–452.

[10] B. Raveh, A. Enosh, and D. Halperin, "A little more, a lot better: Improving path quality by a path-merging algorithm," *IEEE Transactions on Robotics*, vol. 27, no. 2, pp. 365–371, 2011.

[11] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *IJRR*, vol. 26, no. 8, pp. 845–863, 2007.

[12] D. Peleg and A. A. Schäffer, "Graph spanners," *Journal of Graph Theory*, vol. 13, no. 1, pp. 99–116, 1989.

[13] G. Narasimhan and M. H. M. Smid, *Geometric spanner networks*. Cambridge University Press, 2007.

[14] S. Arya, G. Dast, D. M. Mount, J. S. Salowe, and M. Smid, "M.h.m: Euclidean spanners: short, thin, and lanky," in *STOC*. ACM press, 1995, pp. 489–498.

[15] M. Elkin and S. Solomon, "Optimal euclidean spanners: really short, thin and lanky," *CoRR*, vol. abs/1207.1831, 2012.

[16] K. L. Clarkson, "Approximation algorithms for shortest path motion planning (extended abstract)," in *STOC*, 1987, pp. 56–65.
[17] L. Aleksandrov, A. Maheshwari, and J.-R. Sack, "Determining approximate shortest paths on weighted polyhedral surfaces," *J. ACM*, vol. 52, no. 1, pp. 25–53, 2005.
[18] S. Har-Peled, "Constructing approximate shortest path maps in three dimensions," *SIAM J. Comput.*, vol. 28, no. 4, pp. 1182–1197, 1999.
[19] K. R. Varadarajan and P. K. Agarwal, "Approximating shortest paths on a nonconvex polyhedron," *SIAM J. Comput.*, vol. 30, no. 4, pp. 1321–1340, 2000.
[20] J. D. Marble and K. E. Bekris, "Computing spanners of asymptotically optimal probabilistic roadmaps," in *IROS*, 2011, pp. 4292–4298.
[21] A. Dobson, A. Krontiris, and K. E. Bekris, "Sparse roadmap spanners," in *WAFR*, June 2012.
[22] J. D. Marble and K. E. Bekris, "Asymptotically near-optimal is good enough for motion planning," in *ISRR*, 2011.
[23] ——, "Towards small asymptotically near-optimal roadmaps," in *ICRA*, 2012, pp. 2557–2562.
[24] P. K. Agarwal, R. Sharathkumar, and H. Yu, "Approximate euclidean shortest paths amid convex obstacles," in *SODA*, 2009, pp. 283–292.
[25] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner, *Level of Detail for 3D Graphics*. Morgan Kaufman, 2002.
[26] P. K. Agarwal and S. Suri, "Surface approximation and geometric partitions," *SIAM J. Comput.*, vol. 27, no. 4, pp. 1016–1035, 1998.
[27] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Mesh optimization," *SIGGRAPH*, 1993.
[28] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," in *SIGGRAPH*, 1997, pp. 209–216.
[29] H. Edelsbrunner, *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001.
[30] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, 2012.

APPENDIX

We include additional experimental results regarding the different heuristics evaluated. Tables III, IV and V summarize RSEC's results on all scenarios for the FIFO, Compresability and Clearance heuristic, respectively.

Scenario		drift bound original	$d = 0.1$	$d = 0.2$	$d = 0.4$	$d = 0.8$	$d = 0.16$
Easy	average path length	314.8	99.9%	99.5%	98.7%	98.8%	99.4%
	compression factor	1	1.16	1.8	3.5	9.11	23.6
	vertices	5016	87.5%	61.9%	30.5%	11.4%	5.1%
	edges	48866	85.5%	54.3%	28.2%	10.9%	4.1%
Cubicles	average path length	472.4	99.9%	99.7%	99.5%	101.8%	103.6%
	compression factor	1	1.32	2.2	5.62	20.42	52.23
	vertices	5004	78.3%	50.1%	20.0%	6.7%	3.2%
	edges	45217	75.4%	44.3%	17.3%	4.5%	1.6%
Bug trap	average path length	41.8	100%	99.6%	98.1%	95.8%	95.4%
	compression factor	1	1.03	1.58	3.18	9.03	29.5
	vertices	5011	95.7%	69.5%	35.2%	13.0%	4.8%
	edges	66053	97.0%	62.6%	30.9%	10.8%	3.2%
Alpha puzzle	average path length	127.8	99.9%	99.5%	98.6%	98.0%	98.4%
	compression factor	1	1.09	1.73	3.47	8.51	23.76
	vertices	5053	92.2%	63.7%	30.6%	11.8%	4.5%
	edges	62294	91.6%	56.8%	28.5%	17.7%	4.2%

TABLE III
Average path degradation and compression factor of RSEC with respect to the original graph using the FIFO heuristic. The results are averaged over 5 runs and 150 random queries.

Scenario		drift bound original	$d = 0.1$	$d = 0.2$	$d = 0.4$	$d = 0.8$	$d = 0.16$
Easy	average path length	314.8	99.9%	99.4%	98.8%	99.7%	101.1%
	compression factor	1	1.18	1.82	3.43	8.01	16.0
	vertices	5016	8703%	62.1%	32.7%	15.0%	9.1%
	edges	48866	84.4%	53.5%	28.4%	12%	5.7%
Cubicles	average path length	472.4	99.9%	99.4%	99.9%	103.0%	104.5%
	compression factor	1	1.34	2.21	5.21	13.34	29.14
	vertices	5004	77.7%	51.4%	23.6%	10.8%	6.0%
	edges	45217	74.0%	44.0%	18.2%	6.8%	2.9%
Bug trap	average path length	41.8	99.9%	99.7%	97.7%	95.9%	94.8%
	compression factor	1	1.03	1.63	3.18	8.42	17.82
	vertices	5011	95.7%	67.5%	34.8%	14.2%	8.2%
	edges	66053	97.0%	60.2%	30.9%	11.5%	5.2%
Alpha puzzle	average path length	128.2	99.9%	99.6%	99.5%	99.4%	101.6%
	compression factor	1	1.09	1.82	4.28	12.25	31.29
	vertices	5053	92.3%	64.1%	31.3%	12.3%	6.3%
	edges	62294	91.3%	53.6%	22.0%	7.5%	2.7%

TABLE IV
Average path degradation and compression factor of RSEC with respect to the original graph using the Compressibility heuristic. The results are averaged over 5 runs and 150 random queries.

Scenario		drift bound original	$d = 0.1$	$d = 0.2$	$d = 0.4$	$d = 0.8$	$d = 0.16$
Easy	average path length	314.8	99.9%	99.7%	99.7%	101.4%	104.4%
	compression factor	1	1.18	1.89	4.25	11.44	26.27
	vertices	5016	87.3%	63.1%	32.0%	13.8%	8.1%
	edges	48866	84.3%	50.8%	21.8%	7.7%	2.9%
Cubicles	average path length	472.4	100%	99.7%	100.4%	104.0%	106.0%
	compression factor	1	1.36	2.43	6.79	21.73	45.0
	vertices	5004	77.7%	50.7%	22.3%	8.9%	5.6%
	edges	45217	72.7%	39.0%	13.0%	3.7%	1.5%
Bug trap	average path length	41.8	100%	99.8%	98.7%	96.9%	97.5%
	compression factor	1	1.03	1.65	3.94	12.71	41.27
	vertices	5011	95.8%	69.1%	34.9%	12.8%	5.3%
	edges	66053	96.9%	59.1%	24.0%	7.11%	2.0%
Alpha puzzle	average path length	127.8	99.7%	99.2%	98.3%	97.4%	97.4%
	compression factor	1	1.1	1.8	3.73	9.43	24.18
	vertices	5053	91.7%	61.7%	29.0%	11.1%	5.3%
	edges	62294	91.1%	54.8%	26.4%	10.5%	4.0%

TABLE V
Average path degradation and compression factor of RSEC with respect to the original graph using the Clearance heuristic. The results are averaged over 5 runs and 150 random queries.