University of California
Santa Barbara

Geometric Constraint Removal and Related Problems

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Computer Science

by

Neeraj Kumar

Committee in charge:

Professor Subhash Suri, Chair
Professor Divy Agrawal
Professor Daniel Lokshtanov

March 2020

The Dissertation of Neeraj Kumar is approved.

Professor Divy Agrawal

Professor Daniel Lokshtanov

Professor Subhash Suri, Committee Chair

March 2020

Geometric Constraint Removal and Related Problems

by

Neeraj Kumar

To my grandfather

Acknowledgements

I will begin by thanking my advisor Prof. Subhash Suri for introducing me to research in computational geometry and geometric shortest paths and helping me find a path short enough to this dissertation. I am extremly grateful for his technical insights, constant encouragement, invaluable feedback and kindness over the past five years. I would also like to thank Prof. Daniel Lokshtanov for being on my committee and for the research discussions on some of the problems presented in this dissertation. Huge thanks are also due to Prof Divy Agrawal for being on my committee, and to John Hershberger for the technical discussions during his visits to UCSB and his help during the summer of 2018 when I was staying in Portland.

I would also like to thank my collaborators Stavros Sintos for always being available and keen to discuss research, and Sayan Bandyapadhyaya for all the research discussions and friendship during his six-month visit to UCSB.

I am extremely grateful to all my friends who made life in Santa Barbara enjoyable. Special thanks go to Aditya Maheshwari and Nhan Huynh who were my housemates and closest friends for a big chunk of these five years. Thanks are also due to Pratik Soni, Daniil Bochkov, Carol Tsai, Ryan Su, Isaac Mackey and Alex Jones for their friendship, kindness and support. I am also thankful for my friends from Waterloo, specifically Vijay Menon, David Szepesvari and Shreya Agrawal for passing the test of time and still keeping in touch despite being separated by timezones. Special thanks are due to Christina Dee for her support, kindness, humor and being a wonderful friend over all these years.

Finally, I would like to thank my family for years of unconditional love and support – my parents for their hard work and sacrifices, my brother for his love and support, and my grandparents for their blessings. I am forever grateful for all they have done.

Curriculum Vitæ

Neeraj Kumar

Education

2020	Ph.D. in Computer Science, University of California, Santa Barbara.
2015	Master of Mathematics (Computer Science), University of Waterloo, Canada.
2010	Bachelors in Computer Science and Engineering, Indian Institute of Technology, Varanasi, India.

Publications

1. **The Maximum Exposure Problem**
   with Stavros Sintos and Subhash Suri at *22nd International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) 2019, MIT, USA.*
   Submitted to Computational Geometry : Theory and Applications
2. **Computing a Minimum Color Path in Edge Colored Graphs**
   at *Symposium of Experimental Algorithms (SEA) 2019, Kalamata, Greece.*
3. **Improved Approximation Bounds for the Minimum Constraint Removal Problem**
   with Sayan Bandyapadhyaya, Subhash Suri and Kasturi Varadrajan at *21st International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) 2018, Princeton, USA.*
   Also Appears in Computational Geometry : Theory and Applications
4. **Computing Shortest Paths in the Plane with Removable Obstacles**
   with Pankaj K Agarwal, Stavros Sintos and Subhash Suri at *16th Scandinavian Symposium and Wrokshops on Algorithm Theory (SWAT) 2018, Malmo, Sweden.*
5. **Shortest paths in the plane with Violations.**
   with John Hershberger and Subhash Suri at *25th European Symposium of Algorithms, (ESA) 2017, Vienna, Austria.*
   Also Appears in Algorithmica
6. **Counting Convex k-gons in an Arrangement of Line Segments**
   with Martin Fink and Subhash Suri at 28^{th} Canadian Conference on Computational Geometry (CCCG'16), Vancouver, Canada.
7. **SiPTA: Signal Processing for Trace-based Anomaly Detection**
   MM Zeinali, MA Salem, **N Kumar**, G Cutulenco and S Fischmeister, $EMSOFT'14$.

Work Experience

Jun-Sep 2019	Facebook Inc., Cambridge, PhD Intern
Jun-Sep 2018	Intel Corporation, Hillsboro, Graduate Technical Intern
Jun-Sep 2016	Intel Corporation, Santa Clara, Graduate Technical Intern
May 2010 - Aug 2013	Mentor Graphics, India, Senior Member of Technical Staff

Miscellaneous

- **Scholarships and Awards**
  - Distinguished Graduate Student Speaker (UCSB), 2018.
  - Lead Teaching Assistant, Computer Science (UCSB), 2017-18.
  - Outstanding Teaching Assistant (UCSB), 2015-16.
  - Graduate Entrance Scholarship (UWaterloo), 2013.
- **Teaching assistant**
  - *Graduate:* CS 235 (Computational Geometry , UCSB), CS 231 (Advanced Algorithms , UCSB)
  - *Undergraduate:* CS 130A, 130B (Algorithms and Data Structures, UCSB) CS341 (Algorithms, UWaterloo)
- **External Reviewer for Conferences**
  - European Symposium on Algorithms (ESA) 2018
  - Symposium on Computational Geometry (SoCG) 2019
  - Knowledge and Data Discovery (KDD) 2019, 2020
- **External Reviewer for Journals**
  - Information Processing Letters (IPL)
  - Journal of Combinatorial Optimization (JOCO)
  - ACM Transaction on Knowledge Discovery From Data (TKDD)

Abstract

Geometric Constraint Removal and Related Problems

by

Neeraj Kumar

In a geometric optimization problem, the goal is to optimize an objective function subject to a set of constraints induced by a family of geometric objects. When these constraints render the problem infeasible or cause the objective function value to be unacceptable, a natural course of action is to remove or relax some of the constraints, without changing the problem formulation too much. In this dissertation, we study natural formulations of two geometric optimization problems subject to a fixed budget on the number of constraints that can be removed.

For most parts, our focus is on path finding problems in the plane in presence of polygonal obstacles, where we are allowed to remove a small number of the obstacles. We first consider the case when obstacles are overlapping such that no *feasible* path between a given source s and destination t exists. Here, one would like to remove the minimum number of obstacles so that there exists an obstacle-free s–t path. This problem is more commonly known as *minimum constraint removal* (MCR), and is known to be NP-hard, even when obstacles are axis-aligned rectangles. We design approximation algorithms for MCR, which are based on solving the related problem of computing a minimum color path in a *colored graph*.

Next, we consider the case when obstacles are disjoint (so a feasible path always exists) but even the shortest such path is unacceptably long. For this case, we design polynomial-time algorithms to compute a set of at most k obstacles removing which reduces the original shortest path by maximum amount. An important requirement is

that the obstacles must be convex polygons.

Finally, we consider another geometric optimization problem called *maximum exposure*. Here, we are given a set of points P and a set of ranges $\mathcal{R}$ covering them, and we would like to remove a subset $\mathcal{R}'$ of at most k ranges so as to 'expose' (or uncover) a maximum number of points. Our work here deals with designing approximation algorithms.

Contents

Chapter 1

Introduction

In its most generic form, an *optimization problem* refers to the problem of finding the best solution from a space of feasible solutions. Optimization problems are everywhere, from deciding what transit option to take, to financial planning, we instinctively solve optimization problems on a regular basis. Quite naturally, due to its significance to computing, the study of optimization problems continues to be one of the most well-explored areas in computer science. Algorithms for combinatorial optimization problems such as the shortest path in a graph, minimum spanning tree among many others are concepts so fundamental that they have been part of introductory algorithm courses for decades. Moreover, such algorithms have found applications in a large number of areas, such as, networking, robotics, circuit design, machine learning to name a few.

More formally, in an optimization problem, we are given some objective function f, a set of constraints that restrict the space of feasible solutions, and our goal is to find a solution x_{opt} such that the value $f(x_{opt})$ is optimized. Geometric optimization refers to a subclass of these problems where the underlying constraints are induced by a family of geometric objects. As an example, consider the shortest path problem in the plane : given a set of obstacles (modeled as polygons), a source point s and target t, we

want to compute a minimum length path from s to t that does not intersects any of the obstacles. In other words, the obstacle polygons restrict the space of all feasible s-t paths. Linear programming, Euclidean TSP, clustering are some other well-known geometric optimization problems. Over the years, researchers have designed a number of algorithms to efficiently compute the optimal solutions for various geometric optimization problems. However, an equally important question to consider is what happens if there is no feasible solution or if the feasible solution value is unacceptable. Indeed, a natural course of action is to relax or remove some of the constraints, but finding out which constraints to remove turns out to be a challenging optimization problem by itself.

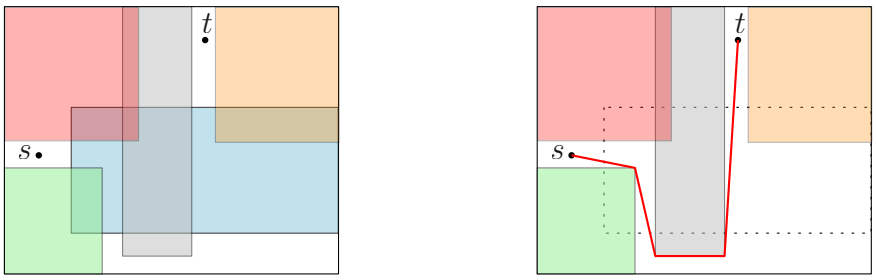
In this dissertation, we consider geometric optimization problems with a fixed budget on the number of constraints we can remove. Our focus is on path finding and exposure problems in the plane where constraints are defined by geometric objects, and therefore removing a constraint essentially means deleting an object from the input. Clearly, every set of objects that are deleted corresponds to an optimal solution value x'_{opt} that optimizes the value $f(x'_{opt})$ in the modified instance. We would like to find the set of objects to delete that optimizes $f(x'_{opt})$ over all such choices. If n is the number of objects and k is the budget, one can indeed try all possible $\binom{n}{k}$ choices, and return the one that optimizes the objective function value. The challenge however is to do this in a computationally efficient way.

1.1 Problems Studied and Contributions

In this section, we present a high level overview of the problems studied in this dissertation and our contributions to them.

1.1.1 Minimum Constraint Removal (MCR)

Suppose we are given a set $\mathcal{S}$ of possibly overlapping obstacles in the plane, and two fixed points s and t. Then depending on the arrangement of obstacles, it is possible that there is no feasible (obstacle-free) path from s to t. In the minimum constraint removal problem, our goal is to find the smallest set of obstacles $\mathcal{S}' \subseteq \mathcal{S}$ to remove so that there is an obstacle-free path from s to t. (See also Figure 1.1).



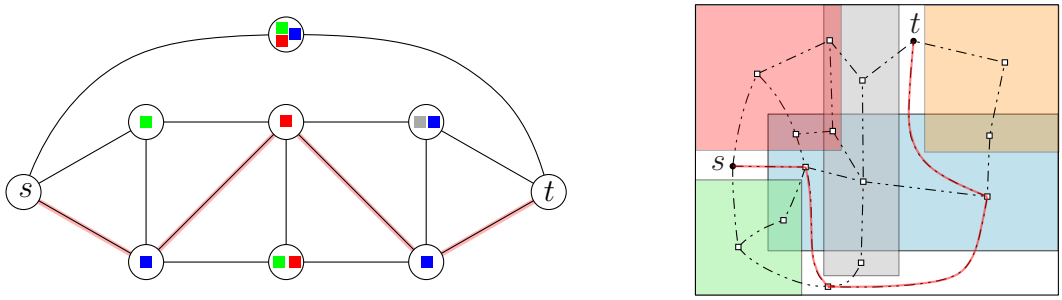
(a) (b)

Figure 1.1: (a) An example of MCR with rectangular obstacles. At least one obstacle must be removed to open a path from s to t. (b) Removing the obstacle shown in dotted is sufficient.

The problem is known to be NP-complete even for simple obstacle shapes, such as convex obstacles [1], line segments [2, 3, 4], or axis-parallel rectangles with aspect ratio close to one [5]. A natural course of action is to consider approximate solutions with provable guarantees. In particular, can we find a set of obstacles $\mathcal{S}'$ to remove such that the number of obstacles in $\mathcal{S}'$ is at most some factor f of the optimal solution?

We show that this is indeed possible, and provide two approximation algorithms: one with approximation factor $f = O(\sqrt{n})$ using relatively simpler techniques; and another with $f = O(\log n)$ using a Linear programming formulation combined with some careful rounding techniques. Both these algorithms build on the following initial observation of translating MCR to a path finding problem on a *colored graph*.

Min-Color Path Suppose we are given a graph $G = (V, E)$ and a set of colors $\mathcal{C}$, such that each vertex v is assigned a set of colors $\chi(v) \subseteq \mathcal{C}$. Now, given two fixed vertices s and t, the goal is to find a path from s to t that minimizes the total number of colors along the path. In other words, we want to find a path π such that $|\bigcup_{v \in \pi} \chi(v)|$ is minimized. (See also Figure 1.2(a).)



(a) $\mathcal{C} = \{$ ■, ■, ■, ■ $\}$ (b)

Figure 1.2: (a) An example of min-color path on a graph with color set $\mathcal{C}$: optimal path that uses two colors is highlighted. (b) Translating an instance of MCR to min-color path : colors in the path corresponds to obstacles that are removed.

MCR to Min-Color Path We can now translate an instance of MCR to an instance of minimum color path problem by constructing a graph on the *arrangement* formed by the obstacles. The arrangement $\mathcal{A}(\mathcal{S})$ of $\mathcal{S}$ is a partition induced by the obstacles, whose *faces* are the two-dimensional connected regions, and whose edges are segments of the obstacle boundaries. We now define a planar graph $G_{\mathcal{A}}$ whose vertices are in one-to-one correspondence with the faces of the arrangement, and whose edges join two neighboring faces. By associating each obstacle with a unique color, we obtain a version of the minimum color path problem : an s–t path has exactly as many colors along it as the number of obstacles it crosses. However, it is worth pointing out that the number of vertices in $G_{\mathcal{A}}$ can be *quadratic* in the size of the geometric input: a set of n geometric obstacles, each with a constant number of boundary edges, can create an $\Omega(n^2)$ size arrangement. (See also Figure 1.2(b).)

In order to design approximation algorithms, we first try to understand the complexity of min-color path on graphs. For this problem, we obtain some hardness bounds, as well as design some non-trivial approximation algorithms. Observe that MCR is equivalent to solving a special case of min-color path : the underlying graph G is planar and the set of vertices containing a given color are connected. As we will later see, that planarity and color-connectivity are precisely the reason why one can design 'good' approximation algorithms for MCR. In all other cases, the problem remains hopelessly hard to approximate.

1.1.2 Shortest Paths with Removable Obstacles

The next problem we consider is quite similar to MCR in the sense that given an arrangement of obstacles $\mathcal{S}$ in the plane, we want to remove obstacles to find a *good path* from s to t but with one important distinction: the obstacles are *convex* and *non-overlapping*. This is motivated from real world applications in robotics, or urban planning, where the obstacles are pairwise disjoint. Indeed, there always exists an s–t path but it could be unacceptably long. Therefore, we want to remove a set of at most k obstacles so that the shortest path length among remaining obstacles is minimized. (See also Figure 1.3).

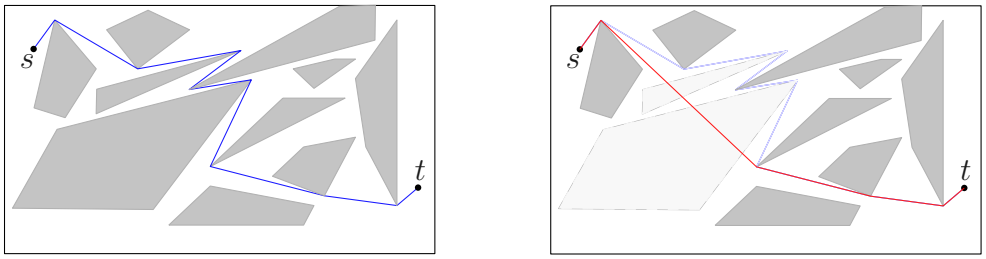


Figure 1.3: An example of shortest path among disjoint removable obstacles in plane. Removing the two highlighted obstacles minimized the shortest path length for $k = 2$.

We show that this problem can be solved in polynomial time. In fact, we give an

algorithm that runs in $O(k^2 n \log n)$ time which is optimal for $k = O(1)$. Next, we consider the case when each obstacle has a non-negative *cost* of removal and we are given a budget C on the total cost of obstacles we can remove. Although solving this variant exactly turns out to be NP-complete, we show that if we relax the cost budget to $(1+\epsilon)C$ for some fixed $\epsilon > 0$, the problem can be solved in time polynomial in n and $1/\epsilon$.

1.1.3 Maximum Exposure

We now move to the final problem we consider in this dissertation called *maximum exposure* which concerns with reliability of coverage of points in the plane by a family of geometric objects. More formally, let $S = (P, \mathcal{R})$ be a geometric set system, also called a *range space*, where P is a set of points and each $R \in \mathcal{R}$ is a subset of P, also called a range. We say that a point $p \in P$ is *exposed* if no range in $\mathcal{R}$ contains p. The *max-exposure* problem is defined as follows: given a range space $(P, \mathcal{R})$ and an integer parameter $k \geq 1$, remove k ranges from $\mathcal{R}$ so that a maximum number of points are exposed. That is, we want to find a subfamily $\mathcal{R}^* \subseteq \mathcal{R}$ with $|\mathcal{R}^*| = k$, so that the number of exposed points in the (reduced) range space $(P, \mathcal{R} \setminus \mathcal{R}^*)$ is maximized. (See also Figure 1.4).

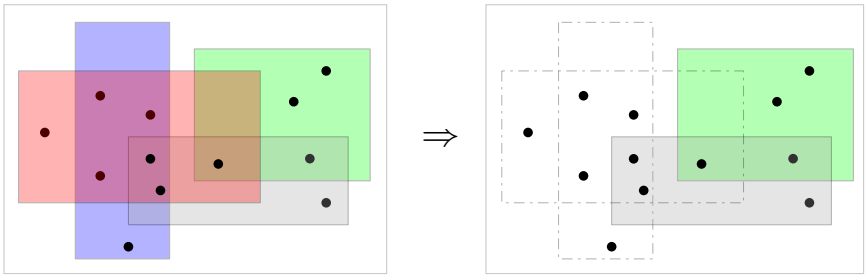


Figure 1.4: An instance of max-exposure with rectangular ranges and its solution for $k = 2$.

We are primarily interested in range spaces defined by a set of points in two dimensions and ranges defined by axis-aligned rectangles. We show that the problem is NP-hard and assuming plausible complexity conjectures is also hard to approximate even when

rectangles in $\mathcal{R}$ are translates of two fixed rectangles. However, if $\mathcal{R}$ only consists of translates of a single rectangle, we present a polynomial-time approximation scheme. For general rectangle range space, we present a simple $O(k)$ bicriteria approximation algorithm; that is by deleting $O(k^2)$ rectangles, we can expose at least $\Omega(1/k)$ of the optimal number of points. The bicriteria approximation bounds hold for any polygon with a constant number of sides, as well as for arbitrary *pseudodisks*. Note that a set of objects is a collection of pseudodisks, if the boundary of every pair of them intersects at most twice.

1.2 Organization of Chapters

The remainder of this dissertation is organized as follows. In Chapter 2, we study the min-color path problem on general graphs in detail. We establish some hardness of approximation guarantees and design sublinear approximation algorithms for both vertex-colored and edge-colored variants. Chapters 3 and 4 deal with approximation algorithms for MCR. We outline the details of an $O(\sqrt{n})$ factor approximation algorithm for MCR in Chapter 3, which is then subsequently improved to an $O(\log n)$-approximation in Chapter 4.

We study the problem of shortest path among removable obstacles in Chapters 5 and 6. In Chapter 5, we design an $O(k^2 n \log n)$ algorithm for this problem. If we consider L_1 (manhattan) distances, we can obtain a faster $O(kn \log^2 n)$ algorithm which is discussed in Chapter 6. This is then also used to obtain a fast $(1+\epsilon)$-approximation algorithm for the variant when each obstacle has a non-negative cost of removal.

Finally, we study maximum exposure in Chapter 7 and conclude with Chapter 8.

1.3 Permissions and Attributions

Most of the research presented in this dissertation have either already appeared in conference proceedings (or journals) or is currently in the process of submission. The specific details on chapters is as follows.

1. The content of Chapters 2 and 4 is mostly based on joint work with D. Lokshtanov and S. Suri and is currently under the process of submission. Parts of Chapter 2 also appear as paper [6] in the proceedings of SEA'2019 published by Springer and available online at `https://doi.org/10.1007/978-3-030-34029-2_3`

2. The content of Chapter 3 is based on joint work with S. Bandyapadhyaya, S. Suri and K. Varadrajan, and parts of it have previously appeared in proceedings of APPROX'2018 as paper [7].

3. The content of Chapter 5 is based on joint work with J. Hershberger and S. Suri, and has been published in Algorithmica. The final authenticated version is available online at `https://doi.org/10.1007/s00453-020-00673-y`

4. The content of Chapter 6 is based on joint work with P.K Agarwal, S.Sintos and S. Suri and has have previously appeared in proceedings of SWAT'2018 as paper [8].

5. The content of Chapter 7 is based on joint work with S.Sintos and S. Suri and has have previously appeared in proceedings of APPROX'2019 as paper [9].

Chapter 2

Minimum Color Path in Graphs

In this chapter, we will study the complexity of computing a min-color path in vertex and edge colored graphs. Formally, a colored graph $G = (V, E, \mathcal{C})$ is called *vertex-colored* if every vertex $v \in V$ is assigned a color set $\chi(v) \subseteq \mathcal{C}$. Similarly, if the colors lie on edges, that is every edge $e \in E$ is assigned a color set $\chi(e) \subseteq \mathcal{C}$, we will refer to it as an *edge-colored* graph. The set of colors used by a path is simply the union of colorsets on the vertices (or edges) of the path and our goal is to compute a *min-color path*, which is a path that uses a minimum number of colors.

The min-color path problem is known to be NP-hard and also hard to approximate within a factor of $o(\log n)$. This follows from a simple reduction from Set Cover. In particular, let $\mathcal{S}$ be a collection of m sets prom an universe $\mathcal{U}$ with n elements., we can construct an instance of min-color path on an edge-colored graph as follows. For each $u_i \in \mathcal{U}$, create a vertex v_i and place them sequentially from left to right in a line. Now, for each set $S_j \in \mathcal{S}$ that contains an element v_i, add an edge (v_{i-1}, v_i) and assign it the color c_j corresponding to $_j$. If we set $s = v_1$ and $t = v_n$, a minimum color path must go through all vertices (cover all elements) and uses a minimum number of unique colors (sets), which is the set cover problem. The graph can easily be made vertex-colored by

adding vertices of degree two for each edge and assigning the color on the edges to the newly added vertex.

Results and Chapter Organization

In this chapter, we focus on two things. First, we work towards designing better hardness of approximation guarantees (lowerbounds). Then, we also work towards designing sublinear approximation algorithms for vertex and edge-colored versions of min-color path. The remainder of this chapter is organized as follows. In Section 2.1, by a reduction from *minimum k-union* problem, we show that min-color path is conditionally hard to approximate within a factor $O(n^{1/8})$ of optimum for both vertex and edge-colored graphs. It is important to note that the hardness of minimum k-union is based on the so-called Dense vs Random conjecture [10] being true. In Section 2.2, we improve this lowerbound to $O(n^{1/4})$ for vertex-colored graphs and $O(n^{1/3})$ for edge-colored graphs by a construction that builds directly on the Dense vs Random conjecture. In Section 2.3, we will discuss an $O(\sqrt{n})$-approximation algorithm for vertex-colored graphs and an $O(n^{2/3})$-approximation algorithm for edge-colored graphs.

2.1 Hardness of Approximation

In the *minimum k-union* problem, we are given a collection $\mathcal{S}$ of m sets over a ground set U and the objective is to pick a sub-collection $\mathcal{S}' \subseteq \mathcal{S}$ of size k such that the union of all sets in $\mathcal{S}'$ is minimized. The problem is known to be hard to approximate within a factor $O(m^{1/4})$ assuming the Dense vs Random conjecture [10]. The conjecture has also been used to give lower bound guarantees for several other problems such as Densest k-subgraph [11], Lowest Degree 2-Spanner, Smallest m-edge subgraph [12], and Label cover [13].

In the following, we will show how to transform an instance of minimum k-union problem to an instance of min-color path on edge-colored graphs. The construction can be easily extended to vertex-colored graphs by adding vertex of degree two for each edge. In particular, given a collection $\mathcal{S}$ of m sets over a ground set U and a parameter k, we will construct an edge-colored graph $G = (V, E, \mathcal{C}, \chi)$ with two designated vertices s, t, such that a solution for min-color path on G corresponds to a solution of minimum k-union on $\mathcal{S}$ and vice versa.

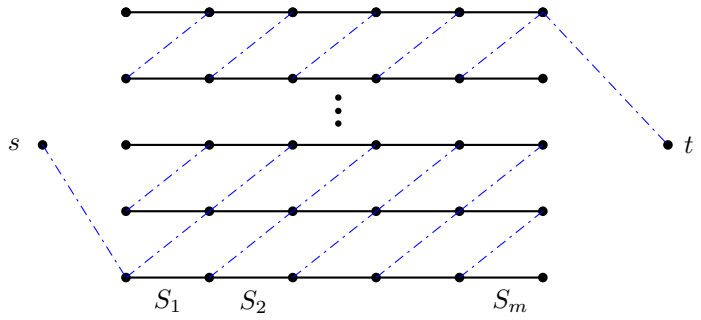


Figure 2.1: Reducing minimum k-union to min-color path. The dashed edges are uncolored. The horizontal edge $(v_{ij}, v_{i(j+1)})$ is assigned color corresponding to the set S_j in all rows i.

We construct G in three steps (See also Figure 2.1).

- We start with a path graph G' that has $m+1$ vertices and m edges. Next, we create $m-k+1$ copies of G' and arrange them as rows in a $(m-k+1) \times (m+1)$-grid, as shown in Figure 2.1.

- So far we only have horizontal edges in this grid of the form $(v_{ij}, v_{i(j+1)})$. Next, we will add *diagonal edges* of the form $(v_{ij}, v_{(i+1)(j+1)})$ that basically connect a vertex in row i to its right neighbor in row $i+1$.

- Finally, we add the vertices s, t and connect them to bottom-left vertex v_{11} and the top-right vertex $v_{(m-k+1)(m+1)}$, respectively.

We will now assign colors to our graph G. For the set of possible colors $\mathcal{C}$, we will use the ground set U and assign every subset $S_j \in \mathcal{S}$ to horizontal edges of G from left to right. More precisely,

- The diagonal edges of G are not assigned any color.
- Every horizontal edge that connects a node in column j to $j+1$ gets assigned the set of color S_j, for all $j \in 1, 2, \ldots, m$. That is $\chi(v_{ij}, v_{i(j+1)}) = S_j$, for all $i \in \{1, 2, \ldots, m-k+1\}$.

We make the following claim.

Lemma 1 *Assuming that minimum k-union problem is hard to approximate within a factor $O(m^{1/4})$ of optimal, the min-color path problem cannot be approximated within a factor of $O(n^{1/8})$, where m is the number of sets in the collection and n is the number of vertices in G.*

Proof: Consider any $s-t$ path π in G. Without loss of generality, we can assume that π is simple and moves monotonically in the grid. This holds because if π moves non-monotonically in the grid, then we can replace the non-monotone subpath by a path that uses the same (or fewer) number of colors. Now observe that in order to get from s to t, the path π must make a horizontal displacement of m columns and a vertical displacement of $m-k$ rows. Since the vertical movement is only provided by diagonal edges and π can only take at most $m-k$ of them, it must take k horizontal edges. If π is the path that uses minimum number of colors, then the sets S_j corresponding to the horizontal edges taken by π must have the minimum size union and vice versa.

Now suppose it was possible to approximate the min-color path problem within a factor $O(n^{1/8})$ of optimal. Then given an instance of minimum k-union, we can use the above reduction to construct a graph G that has $n = O(m^2)$ vertices and run this

$O(n^{1/8})$ approximation algorithm. This will give us a path π that uses at most $O(m^{1/4})r$ colors, where r is the minimum number of colors used. As shown above, r must also be the number of elements in an optimal solution of minimum k-union. Therefore, we can compute a selection of k sets that have union at most $O(m^{1/4})$ times optimal, which is a contradiction. ■

Note that we can also make G vertex-colored : subdivide each horizontal edge e by adding a vertex v_e of degree two, and assign the set of colors $\chi(e)$ to v_e. Observe that since the graph G we constructed above had $O(n)$ edges, the same lower bound also translates to min-color path on vertex-colored graph.

2.2 Improved Hardness of Approximation

The hardness construction used in the previous section was based on a reduction from minimum k-union, which was shown to be hard based on the Dense vs Random conjecture for hypergraphs. In this section, we provide a different lowerbound construction that uses the conjecture directly, and achieves better hardness of approximation guarantees. We begin by setting a few definitions and then formally stating the conjecture.

Definition 1 *A hypergraph $\mathcal{G} = (X, H)$ is a* r-uniform hypergraph *over a set of vertices X if every hyperedge $h \in H$ is a subset of X with cardinality r. In a* random hypergraph *$\mathcal{G}(n, p, r)$, every subset of size r is chosen to be a hyperedge with probability p.*

Indeed, graphs are just 2-uniform hypergraphs and the random graph $G(n, p) = \mathcal{G}(n, p, 2)$. The Dense vs Random conjecture is then stated as follows.

Dense vs Random Given a hypergraph $\mathcal{G}$, constants $0 < \alpha, \beta < r-1$, and a parameter k, we want to distinguish between the following two cases.

1. (RANDOM) An instance of a r-regular hypergraph $\mathcal{G} = \mathcal{G}(n, p, r)$ for $p = n^{\alpha-(r-1)}$. By construction, such a hypergraph has $n^{\alpha+1}$ edges in expectation and has average degree approximately n^{α}.

2. (DENSE) An instance of a r-regular hypergraph $\mathcal{G}$ that is adversarially chosen so that the densest subhypergraph of $\mathcal{G}$ over k vertices has average degree k^{β}.

Conjecture 1 (Dense vs Random [10]) *For all constant r and $0 < \alpha, \beta < r - 1$, sufficiently small $\epsilon > 0$, and for all k such that $k^{1+\beta} \leq n^{(1+\alpha)/2}$, one cannot distinguish between the* dense *and* random *cases in polynomial time with high probability when $\beta < \alpha - \epsilon$.*

Roughly speaking in order to obtain hardness guarantees for our problem using the above conjecture, we will construct two instances of min-color path : one that is defined by the *dense* case and another that is based on *random case* such that the min-color path in the dense case uses fewer colors and random case uses more colors. Suppose, we define the 'distinguishing ratio' d to be the least multiplicative gap between the minimum number of colors on the random and dense instances. Then if there exists an algorithm with an approximation factor significantly smaller than d, we would be able to use it to distinguish between the dense and random cases thereby refuting the conjecture. Formally, we have the following lemma.

Lemma 2 *Suppose we are given the optimum solution values x_d^* on* dense *and x_r^* on* random *instances of min-color path. Let $d = x_r^*/x_d^*$ be the* distinguishing ratio. *Then, if* DENSE VS RANDOM *is true, an approximation ratio significantly better than d is not possible in polynomial time.*

Proof: Suppose an approximation factor $f \ll d$ can be obtained in polynomial time. So in the dense case we can compute a solution with value at most $x_d^* \cdot f$. Note that

this is strictly less than x_r^*, the value of the random solution. Therefore, we can use this approximation algorithm to distinguish the dense and random cases and therefore refute the DENSE VS RANDOM conjecture. ■

We now prove the following bound on the number of vertices in a subhypergraph of $\mathcal{G}(n,p,r)$ which will be used later.

Lemma 3 *Let $\mathcal{G}(n,p,r)$ be a random hypergraph. Then any subhypergraph of $\mathcal{G}$ with q hyperedges contains $\tilde{\Omega}(\min\{q,(q/p)^{1/r}\})$ vertices with high probability, where $\tilde{\Omega}$ ignores logarithmic factors.*

Proof: Define $z = \min\left\{\frac{q\ln\ln n}{3\ln n},\ \left(\frac{q}{ep\ln n}\right)^{1/r}\right\}$, where ln denotes natural logarithm. We will now show that the probability that any subhypergraph with at most z vertices contains q edges is small.

Let H be any subhypergraph of $\mathcal{G}$ with z vertices. The probability that H has q edges is the same as probability of q successes in $N = \binom{z}{r}$ trials. Recall that each trial corresponds to selecting a subset of size r from vertices of H and has success probability p. Therefore, we have:

$$\begin{aligned}
\Pr\left[H \text{ contains } q \text{ edges}\right] &= \binom{N}{q}\cdot p^q\cdot(1-p)^{(N-q)} \\
&\le \left(\frac{eN}{q}\right)^q\cdot p^q && \text{since } (1-p)^{(N-q)}\le 1 \\
&\le \left(\frac{ez^r}{q}\right)^q\cdot p^q && \text{since } N < z^r \\
&\le \left(\frac{ep}{q}\cdot z^r\right)^q \\
&\le \left(\frac{1}{\ln n}\right)^q && \text{since } z \le \left(\frac{q}{ep\ln n}\right)^{1/r} \\
&\le \left(e^{-\ln\ln n}\right)^q \\
&\le \left(e^{-\ln\ln n}\right)^{3z\ln n/\ln\ln n} && \text{since } z \le \frac{q\ln\ln n}{3\ln n}
\end{aligned}$$

$$\leq \quad e^{\ln n^{-3z}} \quad \leq \quad \frac{1}{n^{3z}}$$

Applying union bound over all possible hypergraphs with z vertices, we get probability that any z sized hypergraph contains q edges is at most $n^z \cdot n^{-3z} = 1/n^{2z}$. Union bound over all (at most n) values of z gives us the final probability $1/n^{2z-1} \leq 1/n$ for any $z > 0$.

■

2.2.1 Construction for Vertex-Colored Graphs

Given a hypergraph $\mathcal{G} = (X, H)$, we will construct an instance of min-color path $G = (V, E, \mathcal{C})$ as follows. Let ℓ be a parameter that we will fix later. We will use the notation $adj(h)$ to denote the set of vertices of X adjacent to hyperedge $h \in H$.

- Define the color set $\mathcal{C} = X$, the set of vertices of the hypergraph.
- Add $\ell + 1$ vertices $v_1, v_2, \ldots, v_{\ell+1}$ to G and arrange them sequentially in the plane from left to right. (See also Figure 2.2.)
- Uniformly partition hyperedges H into ℓ groups as $H_1, H_2, \ldots, H_\ell$. That is, every hyperedge is assigned a group with a probability $1/\ell$ independent of other hyperedges.
- For each hyperedge $h \in H_i$, add a vertex v_h and connect it to vertices v_i and v_{i+1}. Assign the colors corresponding to vertices of h in hypergraph $\mathcal{G}$ as $\chi(v_h) = adj(j)$.

We will use $n = |X|$, the number of vertices and $m = |H|$, the number of hyperedges of $\mathcal{G}$. By our construction, we have $|\mathcal{C}| = n$, $|V| = m + \ell$ and $|E| = O(|V|)$. Next, we prove the following lemma.

Lemma 4 *Let $H^* \subseteq H$ be any subset of hyperedges of $\mathcal{G}$ with size q. If $\ell = \frac{q}{3 \ln n}$, then every group $H_i \in \{H_1, H_2, \ldots, H_\ell\}$ contains at least one edge from H^* w.h.p.*

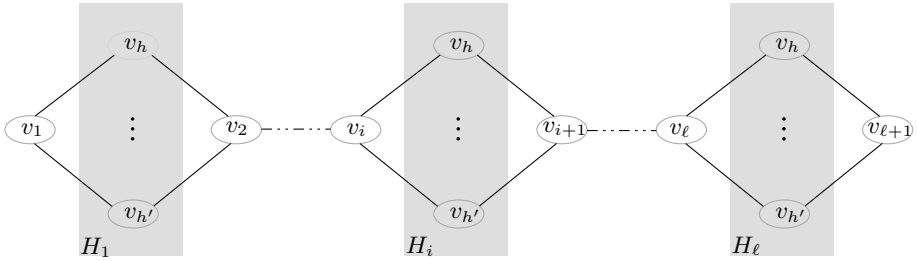


Figure 2.2: An example of the construction. The groups H_1, H_i and H_ℓ are shown shaded in gray.

Proof: This is the same as classic problem of assigning q edges (balls) to ℓ groups (bins). We say that group H_i is H^*-*empty* if $H_i \cap H^* = \emptyset$ (in other words H_i contains no edge from H^*). Let us compute the probability that group H_i is H^*-empty.

$$\begin{aligned} \Pr[\ H_i \text{ is } H^*\text{-empty }] &= \left(1 - \frac{1}{\ell}\right)^q = \left(1 - \frac{1}{\ell}\right)^{\ell \cdot \frac{q}{\ell}} \\ &\leq e^{-\frac{q}{\ell}} = e^{-3\ln n} = \frac{1}{n^3} \end{aligned}$$

The probability that at least one of H_i, for all $i \in \{1, 2, \ldots, \ell\}$, is H^*-empty is at most $\ell/n^3 \leq 1/n$, since $\ell < m \leq n^2$. Therefore, the probability that no group is H^*-empty (contains at least one edge from H^*) is at least $1 - 1/n$. ■

We say that the colored graph G is *dense* if the underlying hypergraph $\mathcal{G}$ was sampled from the dense case. Otherwise we say that G is *random*. Recall that if $\mathcal{G}$ was dense, it contained a dense subhypergraph with $k^{\beta+1}$ edges. We have the following lemma.

Lemma 5 *Let $q = k^{\beta+1}$ and suppose we set the number of groups $\ell = \frac{q}{3\ln n}$ in our construction (See also Figure 2.2). Morever let $s = v_1$ be the source $t = v_\ell$ be the destination. Then the following holds w.h.p :*

1. *min-color s–t path in* dense *case uses at most k colors*

2. *min-color s–t path in* random *case uses $\tilde{\Omega}(\min\{q, (q/p)^{1/r}\})$ colors*

Here $p = n^{1+\alpha-r}$ is the probability of adding hyperedges in $\mathcal{G}(n, p, r)$.

Proof:

1. Since we are in the dense case, we know that $\mathcal{G}$ contains a subhypergraph over k vertices with $k^{\beta+1}$ hyperedges. Let H^* be the set of these hyperedges. Applying Lemma 4, we get that each of the ℓ groups contain at least one edge from H^*. Therefore, the colored graph G contains a path that only uses colors from vertices of the edge set H^*. Since the number of vertices is k, there exists a path in G with at most k colors.

2. Now we are in the random case. The colors on any simple path in the colored graph G correspond to a subhypergraph of $\mathcal{G}$ with ℓ edges. Since hyperedges in $\mathcal{G}$ are added with probability p, applying Lemma 3 on a hypergraph with $\ell = \tilde{\Omega}(q)$ edges, we have that any such path must have $\tilde{\Omega}(\min\{q, (q/p)^{1/r}\})$ colors.

■

From Lemma 5, we have:

$$\begin{aligned} \text{distinguishing ratio} &\geq \frac{\min\{k^{\beta+1}, \quad (k^{\beta+1}/n^{1+\alpha-r})^{1/r}\}}{k} \\ &= \min\left\{k^{\beta}, \quad \left(\frac{k^{\beta+1-r}}{n^{1+\alpha-r}}\right)^{1/r}\right\} \end{aligned} \tag{2.1}$$

Lemma 6 *For the choice of parameters $\alpha = \sqrt{r} - 1$, $\beta = \alpha - \epsilon$ and $k = n^{\frac{1}{\sqrt{r}+1}}$, the distinguishing ratio between dense and random instances is at least $n^{1-\epsilon}$ (w.h.p ignoring log factors) for some arbitrarily small $\epsilon > 0$. Here n is the number of vertices of hypergraph $\mathcal{G}$ which is r-uniform.*

Proof: Substituting the values in Equation 2.1, we get:

$$k^{\beta} = n^{\frac{\sqrt{r}-1-\epsilon}{\sqrt{r}+1}} = n^{1-\frac{2+\epsilon}{\sqrt{r}+1}} = n^{1-\epsilon'} \quad \text{where } \epsilon' = \frac{2+\epsilon}{\sqrt{r}+1}$$

$$\text{Similarly, we have } \left(\frac{k^{\beta+1-r}}{n^{1+\alpha-r}}\right)^{1/r} = n^{\frac{\sqrt{r}-1-\epsilon/r}{\sqrt{r}+1}} = n^{1-\epsilon''} \quad \text{where } \epsilon'' = \frac{2+\epsilon/r}{\sqrt{r}+1}$$

Setting $\epsilon \leftarrow \epsilon'$ gives us the distinguishing ratio to be at least $n^{1-\epsilon}$. Note that this can be made arbitrarily close to n by choosing a significantly large enough r. ■

It is easy to verify that the parameter values in Lemma 1 satisfy the requirements for Conjecture 1. That is, we have $0 \leq \alpha, \beta < 1$ and $k^{\beta+1} = n^{\frac{\sqrt{r}-\epsilon}{\sqrt{r}+1}} \leq n^{1+\alpha}$. Since the number of vertices of $\mathcal{G}$ is the same as number of colors in our construction of G, we have the following theorem.

Theorem 1 *Assuming* Dense vs Random *, min-color path on a colored graph $G = (V, E, \mathcal{C})$ cannot be approximated within a factor significantly better than $O(|\mathcal{C}|)$.*

The above theorem shows that, it is quite unlikely to find a good approximation in terms of the number of colors. We will now obtain a bound in terms of number of vertices of G. Observe that the number of vertices in G is $m + \ell = \Theta(m)$ where m is the number of hyperedges of $\mathcal{G}$.

Therefore, We can rewrite Equation 2.1 by expressing the probability $p = n^{1+\alpha-r}$ from the random hypergraph $\mathcal{G}(n, p, r)$ in terms of the number of edges m. Specifically, we can use $n^{1+\alpha} = m$, which gives $p = m^{(1+\alpha-r)/(1+\alpha)}$.

$$\text{distinguishing ratio} \geq \min\left\{k^{\beta}, \left(\frac{k^{\beta+1-r}}{m^{(1+\alpha-r)/(1+\alpha)}}\right)^{1/r}\right\} \tag{2.2}$$

Similar to Lemma 1, we choose the parameters as follows.

Lemma 7 *For the choice of parameters $\alpha = \frac{r-1}{r+1}$, $\beta = \alpha - \epsilon$ and $k = m^{\frac{r+1}{4r}}$, the*

distinguishing ratio between dense and random instances is at least $m^{1/4-\epsilon}$ (w.h.p ignoring log factors) for some arbitrarily small $\epsilon > 0$. Here m is the number of vertices of hypergraph $\mathcal{G}$ which is r-uniform.

Proof: Substituting the values in Equation 2.2, we get:

$$k^{\beta} = m^{\frac{r+1}{4r}\cdot(\frac{r-1}{r+1}-\epsilon)} = m^{\frac{1}{4}-\frac{1+\epsilon(r+1)}{4r}} = m^{1/4-\epsilon'} \quad \text{where } \epsilon' = \frac{1+\epsilon(r+1)}{4r}$$

Similarly, we have

$$\left(\frac{k^{\beta+1-r}}{m^{(1+\alpha-r)/(1+\alpha)}}\right)^{1/r} = m^{\frac{1}{4}-\frac{1+\epsilon(r+1)/r}{4r}} = m^{1/4-\epsilon''} \quad \text{where } \epsilon'' = \frac{1+\epsilon(r+1)/r}{4r}$$

Setting $\epsilon \leftarrow \epsilon'$ gives us the distinguishing ratio to be at least $m^{1/4-\epsilon}$. Note that this can be made arbitrarily close to $m^{1/4}$ by choosing a significantly large enough r. ■

Again it is easy to verify that the parameter values from Lemma 7 satisfy Conjecture 1. More precisely, we have $0 < \alpha, \beta < (r-1)$ and $k^{\beta+1} < k^{\alpha+1} = m^{\frac{r+1}{4r}\cdot\frac{2r}{r+1}} = \sqrt{m} = n^{\frac{1+\alpha}{2}}$.

Moreover, since the number of vertices in the colored graph G is $\Theta(m)$, we obtain the following theorem.

Theorem 2 *Assuming* Dense vs Random *, min-color path on a colored graph $G = (V, E, \mathcal{C})$ cannot be approximated within a factor significantly better than $O(|V|^{1/4})$.*

Observe that the min-color path instance G that we constructed is *planar*, and in fact has treewidth two.

Color Connectivity A min-color path instance $G = (V, E, \mathcal{C})$ is said to be *color-connected* if the set of vertices corresponding to any given color in $\mathcal{C}$ forms a connected component in G. Indeed, one can make G color-connected by adding a new vertex v^* with the color set $\chi(v^*) = \mathcal{C}$ and connecting it to all other vertices. Note that this only increases treewidth of G by one, but makes the graph non-planar.

Corollary 3 *Assuming* Dense vs Random , *min-color path* $G = (V, E, \mathcal{C})$ *is hard to approximate within a factor significantly better than* $O(|V|^{1/4}$ *even on the following restricted instances:*

1. *G has treewidth two and is* not color-connected

2. *G has treewidth three, is color-connected and is* not planar

The problem can be solved in polynomial-time if G is color-connected and has treewidth two, by a non-trivial dynamic programming algorithm. By a reduction from Vertex Cover, the planar and color-connected case is NP-hard even when treewidth is three. Later we will see that good approximation algorithms exist for planar and color-connected instances of min-color path.

2.2.2 Construction for Edge-Colored Graphs

Since the colored graph that we constructed in previous section has linear number of edges, the same bound of $O(|V|^{1/4}$ also applies to edge-colored graphs. However, it seems likely that one should be able to achieve better bounds by some alternative construction. In this section, we show that a bound of $|V|^{1/3}$ is indeed possible. The construction is quite similar to the vertex-colored graphs with some important differences.

Given a hypergraph $\mathcal{G} = (X, H)$ sampled from dense or random cases, we construct an edge-colored graph $G = (V, E, \mathcal{C})$ as follows.

1. Uniformly partition hyperedges of $\mathcal{G}$ into ℓ groups $H_1, H_2, \ldots, H_\ell$, same as before. The set of colors $\mathcal{C} = X$, the vertices of the hypergraph.

2. Add $\ell + 1$ vertices $v_1, v_2, \ldots, v_{\ell+1}$ from left to right as shown in Figure 2.3. Call them *connecting* vertices.

3. For each group H_i, add a bipartite graph $G_i = (U_i, V_i, E_i)$ such that $|E_i| = |H_i|$ and the number of vertices $|U_i| + |V_i|$ is as small as possible.

4. For the bipartite graph $G_i = (U_i, V_i, E_i)$ corresponding to i-th group, do the following:

   (a) Connect every vertex in U_i to the left connecting vertex v_i. Similarly, connect every vertex in V_i to the right connecting vertex v_{i+1}.

   (b) Uniquely map every hyperedge h from H_i to an edge e of E_i. Assign colors to this edge as $\chi(e) = adj(h)$.

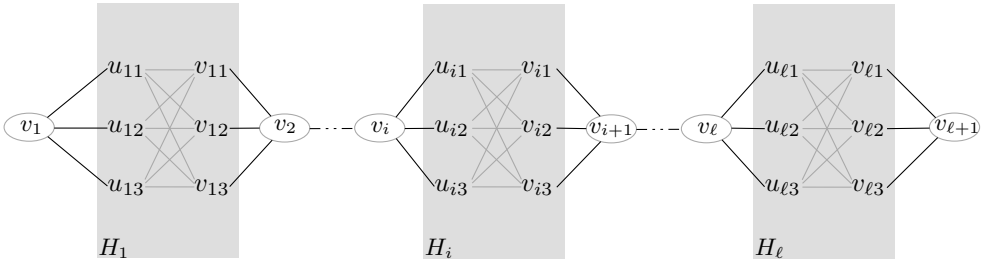


Figure 2.3: An example of the construction. The groups H_1, H_i and H_ℓ are shown shaded in gray. The subgraph G_i in the i-th group is a bipartite graph with $|H_i|$ edges and fewest number of nodes.

Clearly, G is edge-colored. Let $z = |V|$ be the number of vertices of G. Using Conjecture 1, our goal now is to show that min-color path on G cannot be approximated within a factor significantly better than $O(z^{1/3})$. To that end, it suffices to find a choice of parameters α, β, k such that the distinguishing ratio is $\tilde{\Omega}(z^{1/3-\epsilon})$. As usual, we will use n and m to be the number of vertices and edges of the hypergraph $\mathcal{G}$. All calculations are w.h.p and ignore log factors.

We will again apply Lemma 5 with $q = k^{\beta+1}$ and set $\ell = q/3 \ln n$ which guarantees that in the dense case, the min-color path uses at most k colors and in the random case uses $\min\{q, (q/p)^{1/r}\}$ (ignoring log and constant factors). In order to compute the

distinguishing ratio, we need to express the probability p in terms of z. Towards that end, We prove the following lemma.

Lemma 8 *Let the number of groups $\ell = z^{1-x}$ for some $0 < x < 1$, then the probability $p = z^{\frac{(1+\alpha-r)(1+x)}{1+\alpha}}$*

Proof: The expected number of hyperedges in each group is m/ℓ. It follows that the number of edges of the bipartite graph G_i of each group is also be m/ℓ and therefore, the number of vertices of G_i will be $\sqrt{m/\ell}$ w.h.p ignoring log factors. Since the total number of vertices is z, we have that the number of vertices in any given group z^x. This gives us:

$$z^{2x} = \frac{m}{\ell} \implies z^{2x} = \frac{m}{z^{1-x}} \implies m = z^{1+x}$$

Substituting $m = z^{1+x}$ in $p = m^{(1+\alpha-r)/(1+\alpha)}$, we get the claimed value. ■

Moreover, $k^{\beta+1} = \ell = z^{1-x}$. This gives us $k^\beta = z^{\frac{(1-x)\beta}{1+\beta}}$. The distinguishing ratio from Equation 2.2 can now be restated as:

$$\text{distinguishing ratio} \geq \min\left\{ z^{\frac{(1-x)\beta}{1+\beta}}, \left(\frac{z^{(1-x)(1+\beta-r)/(1+\beta)}}{z^{(1+x)(1+\alpha-r)/(1+\alpha)}}\right)^{1/r} \right\} \tag{2.3}$$

Similar to the previous cases, the following lemma can be verified by substituting the parameters to Equation 2.3.

Lemma 9 *For the choice of parameters $\alpha = \frac{r-1}{r+1}$, $\beta = \alpha - \epsilon$ and $x = 1/3$, the distinguishing ratio from Equation 2.3 is at least $z^{1/3-\epsilon}$ (w.h.p ignoring log factors) for some small $\epsilon > 0$.*

Proof: We have $1+\alpha = \frac{2r}{r+1}$, which gives $1+\alpha-r = \frac{r(1-r)}{r+1}$. Moreover, $\frac{\beta}{\beta+1} = \frac{1}{2} - \epsilon'$ for some ϵ' proportional to $1/r$. By similar calculations, $\frac{1+\alpha-r}{1+\alpha} = \frac{1-r}{2}$, and $\frac{1+\beta-r}{1+\beta} = \frac{1-r}{2} - \epsilon''$

for some ϵ'' proportional to $1/r$. Using these values, we get:

$$k^{\beta} = z^{\frac{(1-x)\beta}{1+\beta}} = z^{\frac{2}{3}\cdot(\frac{1}{2}-\epsilon')} = z^{\frac{1}{3}-O(\epsilon')}$$

Similarly, we have $\left(\frac{z^{(1-x)(1+\beta-r)/(1+\beta)}}{z^{(1+x)(1+\alpha-r)/(1+\alpha)}}\right)^{1/r} = \left(\frac{z^{\frac{2}{3}\cdot(\frac{1-r}{2}-\epsilon'')}}{z^{\frac{4}{3}\cdot(\frac{1-r}{2})}}\right)^{1/r} = z^{\frac{1}{3}-O(\epsilon'')}$

■

It remains to see that this choice of parameters satisfies the requirements of Conjecture 1. Clearly, $0 \leq \alpha, \beta < r-1$. Moreover, since $z^{1+x} = m$, we have $k^{\beta+1} = z^{1-x} = m^{(1-x)/(1+x)} = m^{1/2} = n^{(1+\alpha)/2}$. Therefore, we obtain the following theorem.

Theorem 4 *Assuming* Dense vs Random *, min-color path on an* edge-colored *graph* $G = (V, E, \mathcal{C})$ *cannot be approximated within a factor significantly better than* $O(|V|^{1/3})$.

2.3 Approximation Algorithms

We will now describe approximation algorithms to compute the min-color path. More precisely, an algorithm with approximation factor α will compute a path that uses at most $\alpha \cdot k^*$ colors, where k^* is the number of colors used by a min-color path. Although we do not know the value k^* apriori, we know that it is an integer between 1 and $|\mathcal{C}|$, the maximum number of colors. Therefore, we will design an approximation algorithm for computing an approximation for *k-color path* and use it to compute an approximation for min-color path. Note that a *k-color path* is a path that uses exactly k-colors.

An algorithm is called an *α-approximation* algorithm for computing a k-color path if it satisfies the following conditions – if there exists a k-color path, then the algorithm must returns a path with at most αk color, otherwise, the algorithm returns an arbitrary path. The following lemma is straightforward.

Lemma 10 *If there exists an α-approximation algorithm to compute a k-color path then there also exists an α-approximation algorithm for computing a min-color path.*

Proof: We try all possible values $k = 1, 2, \ldots, |\mathcal{C}|$ and let π_k be the path returned by the approximation algorithm for computing a k-color path for a given value of k. Moreover, let π_{k^*} be the path returned by the k-color path approximation algorithm for $k = k^*$. Let j be the value such that $\chi(\pi_j)$ has smallest cardinality over all $\chi(\pi_k)$. Clearly, $|\chi(\pi_j)| \leq |\chi(\pi_{k^*})| \leq \alpha k^*$ and therefore π_j is an α-approximation for computing a minimum color path. ■

From Lemma 10, it follows that computing an approximation of a k-color path is sufficient, and therefore in the rest of our discussion, we work towards that goal.

2.3.1 An $O(\sqrt{n})-$ Approximation for Vertex-Colored Graphs

One natural approach to consider is to simply ignore the colors altogether. Instead, we assign each vertex v a weight equal to $|\chi(v)|$, the cardinality of its color-set and find a minimum weight path using Dijkstra's algorithm. What this does is that we charge for every occurrence of a color on the path. Indeed, if the colors on the path do not repeat at most f times, this simple algorithm computes a f-approximation. However $f = \Theta(n)$, so by itself this does not give a sublinear approximation. But what we can do now is apply a *filtering step*: that identifies and pays for a 'small' number of colors that can occur on the path 'a lot of' times. Now, we know that the colors do not repeat on the path as much, so we can simply apply the Dijkstra's algorithm as usual. The complete details are shown in Algorithm 1. We now make the following claim.

Lemma 11 *Algorithm 1 achieves an $O(\sqrt{n})$-approximation for k-color path on vertex-colored graphs.*

Algorithm 1 Vertex-Colored Graph Approximation

1. Remove all vertices from G that have the number of colors $|\chi(v)| > k$.
2. Discard all colors that occur on at least $\sqrt{n}$ vertices.
3. Set weight of each vertex as $|\chi(v)|$ and return the minimum weight s–t path.

Proof: After all the vertices that contain more than k colors are removed, the total number of occurrences of all colors is bounded by kn. Therefore, the total number of colors that are discarded in Step 2 of the algorithm is at most $kn/\sqrt{n} = k\sqrt{n}$. Finally in Step 3, its easy to see that any color on the minimum weight path can repeat at most $\sqrt{n}$ times, so the k-color path can have weight at most $k\sqrt{n}$. Therefore, the minimum weight path has weight at most $k\sqrt{n}$ in the modified graph and if we include the at most $k\sqrt{n}$ discarded colors, the number of colors used by this path is at most $2k\sqrt{n}$. ■

Combining Lemmas 10 and 11, we obtain the following theorem.

Theorem 5 *Let $G = (V, E, \mathcal{C}, \chi)$ be a vertex-colored graph, then there exists a polynomial time $O(\sqrt{n})$-approximation algorithm for computing a min-color path in G.*

2.3.2 An $O(n^{2/3})-$ Approximation for Edge-Colored Graphs

We now want to compute an approximation algorithm for k-color path on edge-colored graph. We begin with the first natural approach : transform our problem into an instance of k-color path on vertex-colored graphs by adding a vertex of degree two on each edge e and assigning the colors $|\chi(e)|$ to this newly added vertex. Applying Algorithm 1 easily gives an $O(\sqrt{|E|})$-approximation for our problem, which is sub-linear in n if the graph is *sparse*, but can still be $\Omega(n)$ in the worst case. So the challenging case is when the graph is dense.

To address this problem, we apply the technique of Goel et al. [14] where the idea

is to partition the graph G into *dense* and *sparse* components based on the degree of vertices (Step 1 of our algorithm). We consider edges in both these components separately. For edges in dense component, we simply discard their colors, whereas for edges in the sparse components, we use a pruning strategy similar to Algorithm 1 to discard a set of colors based on their occurrence. Finally, we show that both these pieces combined indeed compute a path with small number of colors. We start by making a couple of simple observations that will be useful.

Without loss of generality, we can assume that all edges that contain more than k colors have been removed from G as a k-color path will never use these edges. Since each edge in G now contains at most k colors, we have the following lemma.

Lemma 12 *Any $s-t$ path of length ℓ uses at most $k\ell$ colors and is therefore an ℓ-approximation.*

This suggests that if there exists a path in G of small length, we readily get a good approximation. Note that the diameter of a graph $G = (V, E)$ is bounded by $\frac{|V|}{\delta(G)}$, where $\delta(G)$ is the minimum degree over vertices in G. So if the graph is dense, that is, degree of each vertex is high enough, the diameter will be small, and any path will be a good approximation (Lemma 12).

We are now ready to describe the details of our algorithm. We outline the details for the most general case when the number of colors on each edge is bounded by a parameter $z \leq k$. If z is a constant, the algorithm achieves slightly better bounds.

The input to our algorithm is a colored graph $G = (V, E, \mathcal{C}, \chi)$, two fixed vertices s and t, the number of colors k and a threshold β (which we will fix later) for deciding if a vertex belongs to a dense component or a sparse component. Note that all edges of G have at most $z \leq k$ colors on them.

It remains to show that the algorithm above indeed computes an approximately good

Algorithm 2 Edge-Colored Graph Approximation

1. First, we will classify the vertices of G as lying in sparse or dense component. To do this, we include vertices of degree at most β to the *sparse* component and remove all edges adjacent to it. Now we repeat the process on the modified graph until no such vertex exists. Finally, we assign the remaining vertices to the *dense* component, and restore G to be the original graph.

2. For all edges $e = (u, v)$ such that both u, v lie in the dense component, discard its colors. That is set color $\chi(e) = \emptyset$.

3. Now, consider the set of edges that have at least one endpoint in the sparse component, call them *critical edges*. Note that the number of such edges is at most $n\beta$.

4. Remove every color c_i that occurs on at least $\sqrt{\frac{zn\beta}{k}}$ critical edges. That is, set $\chi(e) = \chi(e) \setminus \{c_i\}$, for all edges $e \in E$.

5. Let G' be the colored graph obtained after above modifications. Using $|\chi(e)|$ as weight of the edge e, run Dijkstra's algorithm to compute a minimum weight $s - t$ path π in G'. Return π.

path. We will prove this in two steps. First, we make the following claim.

Lemma 13 *The number of colors that lie on the path π in the modified colored graph G' is at most $\sqrt{zkn\beta}$.*

Proof: Observe that each color appears on no more than $\sqrt{\frac{zn\beta}{k}}$ edges of G'. Now consider the optimal path π^* in G that uses k colors. Since each of these k colors contribute to the weight of at most $\sqrt{\frac{zn\beta}{k}}$ edges of π^*, the weight of the path π^* in G' is at most $(k \cdot \sqrt{\frac{zn\beta}{k}}) = \sqrt{zkn\beta}$. Therefore, the minimum weight $s - t$ path π will use no more than $\sqrt{zkn\beta}$ colors. ■

Lemma 14 *The number of colors that lie on the path π in the original colored graph G is $O(\frac{zn}{\beta} + \sqrt{zkn\beta})$.*

Proof: To show this, we will first bound the number of colors of π that we may have discarded in Steps 2 and 4 of our algorithm.

Consider a connected dense component C_i. Now let G_i be the subgraph induced by vertices in C_i. Since the degree of each vertex in G_i is at least β, the diameter of G_i is at most $\frac{n_i}{\beta}$, where n_i is the number of vertices in the component C_i. Observe that since the weight of all edges of C_i is zero in G', we can safely assume that π only enters C_i at most once. This holds because if π enters and exits C_i multiple times, we can simply find a shortcut from the first entry to last exit of weight zero, such a shortcut always exists because C_i is connected. Therefore π contains at most $\frac{n_i}{\beta}$ edges and uses at most $\frac{zn_i}{\beta}$ colors in the component C_i. Summed over all components, the total number of colors discarded in Step 2 that can lie on π is at most $z\sum_i \frac{n_i}{\beta} \leq \frac{zn}{\beta}$. Next, we bound the number of colors discarded in Step 4. Observe that since each critical edge contains at most z colors, the total number of occurrences of all colors on all critical edges is $zn\beta$. Since we only discard colors that occur on more than $\sqrt{\frac{zn\beta}{k}}$ edges, the total number of discarded colors is bounded by $\left(zn\beta \Big/ \sqrt{\frac{zn\beta}{k}}\right) = \sqrt{zkn\beta}$.

Summing these two bounds with the one from Lemma 13, we achieve the claimed bound. ■

The bound from Lemma 14 is minimized when $\beta = (\frac{zn}{k})^{1/3}$. This gives the total number of colors used to be $O((\frac{zn}{k})^{2/3}) \cdot k)$ and therefore, an approximation factor of $O((\frac{zn}{k})^{2/3})$. If the number of colors z on each edge is bounded by a constant, we get an approximation factor of $O((\frac{n}{k})^{2/3})$. Otherwise, we have that $z \leq k$, which gives an $O(n^{2/3})$-approximation.

Theorem 6 *There exists a polynomial time $O(n^{2/3})$-approximation algorithm for min-color path in an edge-colored graphs $G = (V, E, \mathcal{C}, \chi)$. If the number of colors on each edge is bounded by a constant, the approximation factor can be improved to $O((\frac{n}{OPT})^{2/3})$.*

2.4 Bibliographic Notes

The min-color path problem was first studied by Yuan et al. [15] and was motivated by applications in maximizing the reliability of connections in mesh networks. More precisely, each network link is assigned one or more colors where each color corresponds to a given failure event that makes the link unusable. Now if the probability of all the failure events is the same, a path that minimizes the number of colors used has also the least probability of failure. Therefore, the number of colors used by a minimum color path can be used as a measure for 'resilience' of the network. This has also been applied in context of sensor networks [16] and attack graphs in computer security [17]. The problem has also gathered significant theoretical interest. If each edge of the graph is assigned exactly one color (called its *label*), the problem is called *min-label* path and was studied in [18]. They gave an algorithm to compute an $O(\sqrt{n})$-approximation and also show that it is hard to approximate within $O(log^c n)$ for any fixed constant c, and n being the number of vertices. Several other authors have also studied related problems such as minimum label spanning tree and minimum label cut [19, 20].

Chapter 3

Minimum Constraint Removal (MCR)

Given a set $\mathcal{S}$ of geometric objects as obstacles in the plane, a path is called *obstacle-free* if it does not intersects the interior of any obstacle. In the *minimum constraint removal* (MCR) problem, the goal is to remove a minimum-sized subset $\mathcal{S}' \subseteq \mathcal{S}$ such that the remaining set $\mathcal{S} \setminus \mathcal{S}'$ admits an obstacle-free path between a source point s and the target point t.

We briefly discussed MCR in Chapter 1 and showed how it could be cast as a minimum color path problem by constructing a graph $G_{\mathcal{A}}$ on the *arrangement* formed by the obstacles. Moreover, we also showed that MCR is a special case of min-color path problem, when the underlying graph is *planar* and the colors are *connected*. In the previous chapter, we studied the minimum color path problem in graphs and gave an $O(\sqrt{n})$-approximation for it on vertex-colored graphs. Indeed, one can try to run this approximation algorithm on the planar graph $G_{\mathcal{A}}$ induced by the arrangement. However, the number of vertices in $G_{\mathcal{A}}$ can be *quadratic* in the size of the geometric input: a set of n geometric obstacles, each with a constant number of boundary edges, can create an

$\Omega(n^2)$ size arrangement. Therefore, applying the approximation algorithm from Chapter 2 directly, only gives an $O(n)$-approximation. In this chapter, we aim to design a sublinear approximation for MCR with geometric objects such as polygons and disks.

Results and Chapter Organization

In particular, we discuss the following results for MCR in this chapter. First, we extend the approximation technique for min-color path from the previous chapter to obtain an algorithmic framework for MCR in Section 3.1. Then we apply the framework in Section 3.2 to obtain an $O(\sqrt{n})$-approximation for rectilinear polygons and pseudodisks. This is later extended to obtain an $O(\sqrt{n\alpha(n)})$-approximation for polygonal objects in Section 3.2.2. Here $\alpha(n)$ is the inverse Ackermann's function and n denotes the total number of vertices of the polygons. We also present an $O(\sqrt{n})$-approximation for rectilinear polygons and pseudodisks.

Finally, we also present some results showing the hardness of approximating MCR. In particular, we show that the problem is NP-hard to approximate within a factor better than 2 for either rectilinear or convex polygons. We also prove the APX-hardness of the problem in a more restricted case, where the obstacles are axis-parallel rectangles. This is discussed in Section 7.1.

Note that in Chapter 4, we will show how to obtain an $O(\log n)$-approximation for MCR by an alternative characterization via colored separators of a planar graph, which significantly improves on the $O(\sqrt{n})$-approximation presented in this chapter. Nevertheless, the algorithms presented in this chapter are relatively simpler and the ideas presented here might be useful in designing faster practical algorithms.

3.1 An Approximation Framework

Roughly speaking, the approximation framework presented here is an abstraction of the simple strategy of removing a 'frequently occurring' color from the graph that gave us an $O(\sqrt{n})$-approximation for vertex-colored graphs in Chapter 2. However, as we will see later, this abstraction makes it easier to obtain better bounds for geometric objects as one just needs to prove some combinatorial properties of the objects in order to obtain a good approximation algorithm.

We are given a colored graph $G = (V, E, \mathcal{C})$ and an integer k, and our goal is develop an approximation algorithm for computing a k-color path in G. (Recall from Lemma 10, that an α-approximation for k-color path gives an α-approximation for min-color path.) The key idea behind our approximation framework is to define a notion of *neighborhood* for the colors in $\mathcal{C}$, and 'discard' the colors that have dense neighborhoods.

Definition 2 *Let $\mathcal{P}$ be an arbitrary set of objects and β be a parameter. We define* neighborhood $\mathcal{N} : \mathcal{C} \to 2^{\mathcal{P}}$ *to be a mapping from $\mathcal{C}$ to subsets of $\mathcal{P}$ that satisfies the following properties.*

1. (**Bounded-Size Property**) *The size of $\mathcal{N}$, defined to be the sum of cardinalities of all neighborhoods, $\sum_{C \in \mathcal{C}} |\mathcal{N}(C)|$, is $O(k\beta^2)$*

2. (**Bounded-Occurrence Property**) *If there exists a k-color path in G, then there also exists a k-color path π^* in G such that, for any color $C \in \mathcal{C}$, the number of times that C appears on π^* is at most $O(|\mathcal{N}(C)|)$.*

The set $\mathcal{P}$ in the above definition can be any set of objects. For example, in min-color path problem $\mathcal{P}$ is the set of vertices of the graph. In the case of MCR, $\mathcal{P}$ is a set of points in the plane. We now describe our approximation algorithm which we will refer to as Approx-Core.

Algorithm 3 APPROX-CORE

1. Construct the neighborhood $\mathcal{N}(C)$ for each color $C \in \mathcal{C}$.
2. For all $C \in \mathcal{C}$, remove all occurrences of the color C from the graph G if $|\mathcal{N}(C)| \geq \beta$. Let G' be the modified graph after removing all such colors.
3. For every vertex v in G', assign an integer weight $|\chi(v)|$ on v.
4. Compute a minimum weight path π from s to t in G' using Dijkstra's Algorithm. Return π.

Lemma 15 *Given the set $\mathcal{P}$ and a parameter β, the algorithm APPROX-CORE gives an $O(\beta)$-approximation for the k-color path in G.*

Proof: Assume that there exists a k-color path in G. Otherwise, the proof is trivial as the algorithm always returns a path. Let $\mathcal{C}_1$ be the set of colors removed during step 2 of the algorithm, and $\mathcal{C}_2$ be the set of colors in G' that appear on the path π returned by the algorithm. Then, the total number of colors in G that may appear on π is at most $|\mathcal{C}_1| + |\mathcal{C}_2|$.

First we compute a bound on the size of $\mathcal{C}_1$. Observe that the neighborhood of each color $C \in \mathcal{C}_1$ has size at least β. Therefore, we have:

$$\begin{aligned} \sum_{C \in \mathcal{C}_1} |\mathcal{N}(C)| &\leq \sum_{C \in \mathcal{C}} |\mathcal{N}(C)| & \\ \implies |\mathcal{C}_1| \cdot \beta &\leq O(k\beta^2) & \text{By the bounded-size property of } \mathcal{N} \\ \implies |\mathcal{C}_1| &\leq ck\beta & \text{for some constant } c \end{aligned}$$

Next, we compute a bound on size of $\mathcal{C}_2$. Towards this end, observe that the neighborhood $\mathcal{N}(C)$ of every color C in G' has fewer than β colors. By the bounded-occurrence property of the neighborhood $\mathcal{N}$, there exists a k-color path π^* in G such that for each $C \in \mathcal{C}$, the number of times that C appears on π^* is at most $O(|\mathcal{N}(C)|)$.

Therefore, it follows that any color C in G' appears on π^* at most $O(\beta)$ times. In other words, there exists a path in G' that has weight at most $c'k\beta$ for another constant c'. Therefore the number of colors used by the minimum weight path π is at most $c'k\beta$.

Hence, the total number of colors in $\mathcal{C}$ that appear on π is at most $|\mathcal{C}_1|+|\mathcal{C}_2| = (c+c')\cdot k\beta$, which is an $O(\beta)$-approximation. ■

We obtain the following theorem.

Theorem 7 *Given a colored graph $G = (V, E, \mathcal{C})$ and integer k, suppose a neighborhood $\mathcal{N}$ for G can be constructed in polynomial time that satisfies the bounded-size and bounded-occurrence property. Then there exists a polynomial time algorithm that achieves an $O(\beta)$-approximation for computing a k-color path in G.*

Therefore, in order to achieve an approximation for the k-color path, it just suffices to construct a neighborhood $\mathcal{N}$, that satisfies the bounded-size and bounded-occurrence properties. In the next section, we illustrate this construction for min-color path on vertex-colored graphs.

3.1.1 Application to Minimum Color Path

In this section, we will apply the above framework to recreate $O(\sqrt{n})$-approximation for min-color path on a vertex-colored graph $G = (V, E, \mathcal{C})$ with n vertices. Our goal is to simply compute a neighborhood $\mathcal{N}$ for a k-color path in G such that $\mathcal{N}$ has bounded-size $O(kn)$ and satisfies the bounded-occurrence property. Using Lemma 10 and $\beta = \sqrt{n}$ in Theorem 7, an $O(\sqrt{n})$-approximation follows.

We define neighborhood $\mathcal{N}(C)$ of each color C to be the set $\{v \in V \mid C \in \chi(v) \text{ and } |\chi(v)| \leq k\}$. The bounded-occurrence property is easily satisfied because a k-color path π_k will never visit vertices that contain more than k colors, and since π_k is

simple, each occurrence of a color C on the path can be uniquely charged to a vertex in $\mathcal{N}(C)$. To see that the bounded-size property is satisfied, we note the following.

$$\sum_{C \in \mathcal{C}} |\mathcal{N}(C)| \;=\; \sum_{v \in V : |\chi(v)| \leq k} |\chi(v)| \;\leq\; kn.$$

Application to Minimum Label Path. As another example application for the framework, we consider a special case of min-color path when each edge has exactly one color (called its label). This problem has been well studied [18, 19, 21] under the name *minimum label path*. Hassin et al. [18] gave an $O(\sqrt{n})$-approximation for this problem on general graphs Using our framework and the following simple definition of neighborhood, we can achieve an $O(\sqrt{\frac{n}{OPT}})$-approximation if the number of edges in G is $O(n)$. Here OPT is the number of labels used by any minimum label s-t path.

For the sake of applying the framework, we transform the input edge-colored graph $G = (V, E, \mathcal{C})$ into a vertex-colored graph H by adding a vertex corresponding to each edge that subdivides the edge. The color corresponding to an old edge is moved to the new vertex. Now, for each new vertex v that has color C, we include both neighbors (old vertices) of v in H to the neighborhood of C. The bounded-occurrence property is straightforward. For the bound on size, observe that an old vertex v can be in at most *degree*(v) neighborhoods, so sum of cardinality of all neighborhoods is at most $2|E|$. Since $|E| = O(n)$, the size of $\mathcal{N}$ is $O(n) = O(\frac{n}{k} \cdot k)$. With $\beta = \sqrt{n/k}$, Theorem 7 and Lemma 10 give an $O(\sqrt{\frac{n}{OPT}})$-approximation.

3.2 Application to Geometric Objects

We now show how to approximate the MCR problem when the obstacles are geometric objects such as constant-complexity polygons or disks. Recall that the approximation

for vertex-colored graphs cannot be applied directly to the graph of the arrangement of obstacles since the latter can have size $\Theta(n^2)$. Instead, we first construct a colored graph G such that an s–t path in the plane that removes the minimum number of obstacles corresponds to a path in G that uses the minimum number of colors. We then construct the neighborhood $\mathcal{N}$ for colors in G such that it satisfies the bounded-size and bounded-occurrence properties. For technical reasons, the graph G we construct for the geometric instances has colors assigned on edges—one can easily transform it into a vertex-colored graph by adding a vertex corresponding to each edge.

Throughout this section, we assume that the obstacles are in *general position*, namely, no three obstacle boundaries intersect at a common point, and the boundaries of any two objects intersect transversally.

Any arrangement of obstacles in the plane can be partitioned into two distinct regions namely the obstacles, and *free space*, that is the region of the plane not occupied by obstacles. Without loss of generality, we assume that the points s and t lie in free space, as we must remove all the obstacles that are incident to either s or t in order to find an obstacle free s–t path. We say that a path π *crosses* an obstacle S if π intersects the interior of S. Note that, as s and t lie in free space, if π crosses S, π must intersect the boundary of S transversally.

Consider an optimal path π that removes the minimum number of obstacles. It is easy to see that π will cross an obstacle S if and only if S was removed from input. Therefore, removing an obstacle is equivalent to crossing it. In the following, we introduce the notion of a *k-crossing path*.

Definition 3 *A path π in the plane is called a* k-crossing path *if it crosses exactly k obstacles.*

It is easy to see that if each obstacle is assigned a unique color and we assign color

to a path whenever it enters an obstacle, then a k-crossing path π uses exactly k colors. Observe that although the space of k-crossing paths is infinite, we want to establish a one to one correspondence between the path in the plane that crosses minimum number of obstacles and a path in G that uses the minimum number of colors.

Towards this end, we simply let G to be the "dual" graph of $G_{\mathcal{A}}$ induced by the input arrangement $\mathcal{A}$: each cell C_i of $G_{\mathcal{A}}$ is associated with a vertex v_i of G that is contained in C_i, and any pair of neighboring cells C_i, C_j are joined by the edge v_iv_j that only intersects the shared boundary ∂C_{ij} between the cells C_i and C_j. Note that the edge v_iv_j is not necessarily a straight line segment, it could be a curve segment in some cases. Due to the general position assumption, ∂C_{ij} is part of the boundary of a unique obstacle in $\mathcal{S}$. Therefore, we have that each edge of G intersects the boundary of a unique obstacle and no two obstacles share an edge. Additionally, we make G directed by replacing each edge $\{v_i, v_j\}$ with two directed edges $v_i \to v_j$ and $v_j \to v_i$. Next, we assign colors to edges of G. Each obstacle in $\mathcal{S}$ corresponds to a color in $\mathcal{C}$, so for any edge $e = v_i \to v_j \in E$, we assign to e the set of colors corresponding to all obstacles S such that v_j lies in the interior of S and v_i does not lie in the interior of S. From our general position assumptions, it follows that $|S|$ is either 1 or 0. Roughly speaking, we assign a color when the edge *enters* into the corresponding obstacle.

Note that the way G is defined, it is a plane graph and we consider its natural embedding which is also planar. Since we assign colors when an edge of G enters an obstacle, it is easy to see that a k-color path π in G corresponds to a k-crossing path π' in the plane. For the other direction, without loss of generality we can assume that there exists a k-crossing path π' that visits a cell in the arrangement at most once. This holds because if not, we can always find a shortcut between two consecutive visits to the same cell. Therefore given such a path π' we can easily construct a path π in G by simply concatenating the vertices corresponding to each arrangement cell intersected by π' in

order. Thus, we have the following immediate observation.

Lemma 16 *Given a set $\mathcal{S}$ of obstacles in the plane, we can build an edge colored graph $G = (V, E, \mathcal{C})$ with two fixed vertices v_s, v_t such that:*

1. *if there is a k-color v_s-v_t path in G, then there is also a j-crossing s-t path in the plane for some $j \leq k$, and*

2. *if there is a k-crossing s-t path in the plane, then there is also a j-color path from v_s to v_t in G for some $j \leq k$.*

We refer to such a graph $G = (V, E, \mathcal{C})$ as a *valid* edge colored graph for the arrangement. From the above discussion and using Lemma 10 and Theorem 7, we have the following.

Lemma 17 *Suppose we are given a valid edge colored graph $G = (V, E, \mathcal{C})$ for an arrangement of the set $\mathcal{S}$ of input obstacles in the plane. Moreover, given integer k, suppose we can construct the neighborhoods $\mathcal{N}(S)$ for all obstacles $S \in \mathcal{S}$ in polynomial time such that $\mathcal{N}$ has a total size of $O(k\beta^2)$ and also satisfies the bounded-occurrence property, where β is independent of k. Then, there exists a polynomial time algorithm that achieves an $O(\beta)$-approximation.*

In the following, given a set $\mathcal{S}$ of obstacles in the plane, we will show how to build a sparse neighborhoods $\mathcal{N}(S)$ for all $S \in \mathcal{S}$, such that the term β^2 in total size of the neighborhoods is close to linear in n.

3.2.1 Building a Sparse Neighborhood

Consider an edge $e = v_i \rightarrow v_j$ of G. We say that e *enters* the obstacle S if e intersects the boundary of S and the cell C_j corresponding to the destination vertex v_j of e is

contained in S. This suggests a natural way of defining the neighborhood $\mathcal{N}(S)$: we simply *include all edges e that* enter *the obstacle S in the neighborhood $\mathcal{N}(S)$ of S*. It is easy to see that every occurrence of the color corresponding to an obstacle S can be uniquely charged to an edge in $\mathcal{N}(S)$. Therefore $\mathcal{N}$ satisfies the *bounded-occurrence* property.

Note that *every edge of the arrangement* is part of the boundary of a cell and due to general position assumption, belongs to the boundary of exactly one obstacle. Therefore, *every edge of our graph G* intersects the boundary of exactly one obstacle, and thus we have the following observation.

Observation 1 *Every edge of G is included in the neighborhood of at most one obstacle.*

For the *bounded-size* property, by Observation 1, we have that the total neighborhood size is $|E|$. However, $|E|$ can be $\Omega(n^2)$, so this directly is not useful for us. Therefore, the primary challenge is to sparsify this neighborhood and in the rest of the discussion, we work towards this goal.

One approach is to only consider the "shallow" edges $e = v_i \to v_j$ such that the destination cells C_j have depth at most k (that is, C_j is contained in at most k obstacles). That is, we define the neighborhood $\mathcal{N}(S)$ of obstacle S to be the set of all shallow edges that enter S. The *bounded-occurrence* property still holds because a k-crossing path cannot enter cells of depth more than k. By planarity, the number of shallow edges is within a multiplicative constant of the number of cells with depth at most k. Indeed, if obstacles are simpler shapes such as disks or pseudodisks, the number of cells with depth at most k is known to be $O(kn)$ [22]. Thus, the total neighborhood size is $O(kn)$. Hence, by Lemma 17, with $\beta = \sqrt{n}$, we readily obtain an $O(\sqrt{n})$-approximation for MCR when the obstacles are pseudodisks.

Theorem 8 *There exists an $O(\sqrt{n})$-approximation for MCR when the obstacles are pseudodisks.*

However, if the obstacles are more general such as rectangles, similar bounds on the number of cells with depth at most k do not hold. Nevertheless, we will use a different notion, called the *level* of a cell, to obtain a more refined neighborhood for the obstacles.

3.2.2 Approximation for Polygonal Objects

In this section, we obtain similar approximation bounds when obstacles are polygonal. We begin by formally defining *level* of a cell.

Definition 4 *The* level of a cell C *in an arrangement, denoted by $L(C)$, is the minimum number of objects one needs to cross to reach C from the source s by a path in the plane. For a collection of cells, we say they are at level at most k if the level of each cell in this collection is at most k.*

With the above definition in place, we can use it to construct the neighborhood $\mathcal{N}$. Recall that the neighborhood $\mathcal{N}(S)$ of an object S (in our earlier definition) consists of all edges $e = v_i \to v_j$ of the graph G that enter the obstacle S into a "shallow" cell C_j. We can now simply use the level of a cell to define its shallowness. Specifically, we *include the edge e to the neighborhood $\mathcal{N}(S)$ if e enters S and the level of its source cell C_i and destination cell C_j are both at most k.* Since a k-crossing path would never enter a cell that has level more than k, bounded occurrence property is easily satisfied. Next, we need to bound the total size of this neighborhood.

Towards that end, suppose we define the level of a vertex of the arrangement $\mathcal{A}(\mathcal{S})$ in the same way. That is, the level of a vertex of the arrangement $\mathcal{A}(\mathcal{S})$ is the minimum number of obstacles crossed by any path from s to the vertex. We define $V_l(n)$ to be the

maximum number of level l vertices in an arrangement of polygons with *input complexity* n. Therefore, $V_0(n)$ upper bounds the number of vertices in $\mathcal{A}(\mathcal{S})$ that can be reached from s without crossing any obstacle. It is important to note that n here is the input complexity (total number of polygon vertices) and not the number of edges of the arrangement, which could be quadratic in n. We now need a technical definition.

Definition 5 *A function $f : \mathbb{Z}^+ \to \mathbb{R}^+$ is* well-behaved *if for any $n \geq 1$ and any random variable X with support $\{0, 1, 2, \ldots, n\}$ and expectation n/m, where $m \geq 1$, we have*

$$E[f(X)] \leq f(n)/m.$$

We will require that $V_0(n)$ is upper bounded by some well-behaved function. Examples of well-behaved functions are n, $n \log n$, and n^2. We now upper bound the number of cells with level at most k, using standard machinery [22].

Lemma 18 *Suppose for a class T of objects, $V_0(n) \leq f(n)$ for some well-behaved function f. Then for any $k \geq 1$, the number of cells with level at most k in an arrangement of the objects from T with input complexity n is $O(k \cdot f(n))$.*

Proof: Let U_k be the set of vertices in the arrangement $\mathcal{A}(\mathcal{S})$ that are at level at most k. We will prove that $|U_k| = O(k \cdot f(n))$. Observe that for every cell that has level at most k, all vertices on its boundary must also be at level at most k. This holds because every point inside the cell is contained in the same set of obstacles, so if we can reach the cell by crossing at most k obstacles, we can also reach the boundary vertices by crossing no more than k obstacles. Therefore by planarity, the number of cells with level at most k will also be bounded by $O(k \cdot f(n))$.

We will now bound the number of vertices in the set U_k. Our approach is based on the well-known probabilistic method due to Clarkson and Shor [22]. Specifically, we sample

each object in $\mathcal{S}$ independently with a probability $p = 1/m$, where $m = 2k$. Let $\mathcal{S}' \subseteq \mathcal{S}$ be the set of sampled objects. Fix one of the vertices $v \in U_k$, and assume its level is $l \leq k$; fix a path from s to v that crosses l objects. There are two cases: either v is an intersection point of two obstacle polygons S_i, S_j or v is a corner of some obstacle S_i. In the first case, v will show up as a level 0 vertex in $\mathcal{A}(\mathcal{S}')$ if $S_i, S_j \in \mathcal{S}'$ and none of the l obstacles from $\mathcal{S}$ that 'block' the path from s to v are sampled in $\mathcal{S}'$. This happens with a probability $p^2(1-p)^l \geq p^2(1-p)^k$. In the second case, v is corner of some polygon S_i. Similar to the earlier argument, v shows up as a level 0 vertex in $\mathcal{A}(\mathcal{S}')$ only if $S_i \in \mathcal{S}'$ and none of the l blocking obstacles in $\mathcal{S}$ are chosen to be in $\mathcal{S}'$. This happens with a probability at least $p(1-p)^k$. By linearity of expectation, the expected number of vertices of U_k that show up as a level 0 vertex in $\mathcal{A}(\mathcal{S}')$ is at least $|U_k|p^2(1-p)^k = |U_k|\left(\frac{1}{m}\right)^2\left(1-\frac{1}{m}\right)^k \geq \alpha|U_k|/k^2$, for some constant $\alpha > 0$.

Now we will upper bound the expected number of level 0 vertices in $\mathcal{A}(\mathcal{S}')$. Since each polygon is sampled with an independent probability of $1/m$, if n was the number of vertices of polygons in $\mathcal{S}$, the expected number of vertices of polygons in $\mathcal{S}'$ is n/m. As f is well-behaved, the expected number of level 0 vertices in $\mathcal{A}(\mathcal{S}')$ is bounded above by $f(n)/m = f(n)/2k$.

It follows that $\alpha|U_k|/k^2 \leq f(n)/2k$. This gives $|U_k| = O(k \cdot f(n))$ and therefore we achieve the claimed bound. ■

Combining Lemma 18 with Lemma 17, we get the following approximation for MCR.

Theorem 9 *Suppose for a class of objects, the maximum number $V_0(n)$ of level 0 vertices is upper bounded by $f(n)$ for some well-behaved function f. Then, there exists an $O(\sqrt{f(n)})$-approximation for MCR with this class of objects.*

From the work due to Edelsbrunner et al. [**?**] it follows that, for arbitrary n segments, $V_0(n) = O(n\alpha(n))$, where $\alpha(n)$ is the functional inverse of Ackermann's function. More-

over, if the segments are axis-parallel, it follows that $V_0(n) = O(n)$ [**?**, 23]. It is easy to see that both upper bounds are in terms of well-behaved functions. For polygonal objects with input complexity n, the number of underlying segments is also $O(n)$, so we obtain the same bounds. In particular, for general polygons $V_0(n) = O(n\alpha(n))$ and for rectilinear polygons, we have $V_0(n) = O(n)$. Hence, we have the following corollary.

Corollary 10 *There exists an $O(\sqrt{n\alpha(n)})$-approximation for MCR with polygonal obstacles. This improves to an $O(\sqrt{n})$-approximation when the polygons are rectilinear.*

3.3 Hardness of Approximation

In this section, we describe the 2-inapproximability and the APX-hardness results for rectilinear polygons and axis-parallel rectangles, respectively.

3.3.1 2-Inapproximability for Rectilinear Polygons

We reduce Vertex Cover to MCR with rectilinear polygons. Recall that as input to the Vertex Cover problem we are given a graph $G = (V, E)$ with n vertices, and the goal is to find a minimum size subset $V' \subseteq V$ such that for any $(u, v) \in E$, either u or v is in V'. Let $e_1, \ldots, e_m$ be the edges of G. Now we describe the reduction. The constructed instance of MCR contains a region called *barrier* formed by a subset of the obstacles. Each point in the barrier is contained in at least $2n$ obstacles and thus if an s-t path intersects the barrier, it intersects at least $2n$ obstacles. We would ensure that any optimal path of the instance intersects at most n obstacles and thus no such path intersects the barrier region. Intuitively, the barrier region forces any optimal path to lie in a certain region, which we refer to as *corridor*.

The construction is the following. We place an obstacle corresponding to each vertex.

For each edge (u, v) there is two possible pathlets (or subpaths of an s-t path) - one that intersects the obstacle corresponding to u and the other that intersects the obstacle corresponding to v. The start points of the two pathlets are same. The end points of the two pathlets are also same. Moreover, the m pairs of pathlets corresponding to the edges are placed one after the other in a series. This ensures that the endpoints of a pair are same as the start points of the next pair. Note that the selection of the pathlet corresponding to u (resp. v) for making an s-t path is equivalent to the selection of u (resp. v) for covering the edge (u, v). Thus, the s-t path formed by the chosen pathlets intersects only k obstacles if and only if all the edges can be covered by k vertices.

Next, we give more details about the layout of the pathlets and the obstacles. One of a pair of pathlets corresponding to each edge lies above x-axis and the other lies below x-axis. To ensure this, all the start and the end points of the pathlets are placed on the x-axis. Let s_i and t_i be the respective start and end points of the pathlets corresponding to the edge e_i. These points are placed on x-axis in the order $s_1, t_1, s_2, t_2, \ldots, s_m, t_m$. For each $1 \leq i < m$, we connect the point t_i with s_{i+1} using a segment that joins the i^{th} and $i + 1^{th}$ pathlets. The point s is placed on x-axis before s_1 and t is placed on x-axis after t_m. s and s_1 are connected by a segment. Similarly, t_m and t are connected by a segment. Now to ensure that the pathlets cross the correct obstacles they are laid out in a fashion as shown in Figure 3.1. Each pathlet contains exactly one point (tip) having the maximum x-coordinate. Moreover, all such tips corresponding to the pathlets are in convex position. Thus one can connect the tips of any subset of pathlets using segments to form a rectilinear polygon that does not intersect any other pathlets (see Figure 3.1). Recall that each pathlet of an edge corresponds to a vertex incident on that edge. For each vertex $u \in V$, we connect the tips of the pathlets corresponding to u to form an obstacle whose shape is a rectilinear polygon. Note that the total number of possible s-t paths we constructed is 2^m. Now to make sure that any optimal s-t path is one of

these 2^m paths we place the barrier around these paths. In other words, the barrier region forms a corridor for the paths. Any optimal path always stays inside the corridor, as it is expensive to cross the "wall" of the barrier. As the pathlets consist of a polynomial number of segments in total, a polynomial number of rectilinear polygons is sufficient to place avoiding the 2^m s-t paths. We make $2n$ copies of each such polygon to ensure the density. Lastly, each obstacle corresponding to a vertex is expanded sufficiently to ensure that it blocks the respective portion of the corridor. Note that the barrier can be placed in a way so that the corridor is arbitrarily thin, and thus this expansion can be done such that the obstacle do not cross any additional pathlets. From the above discussion, we obtain the following lemma.

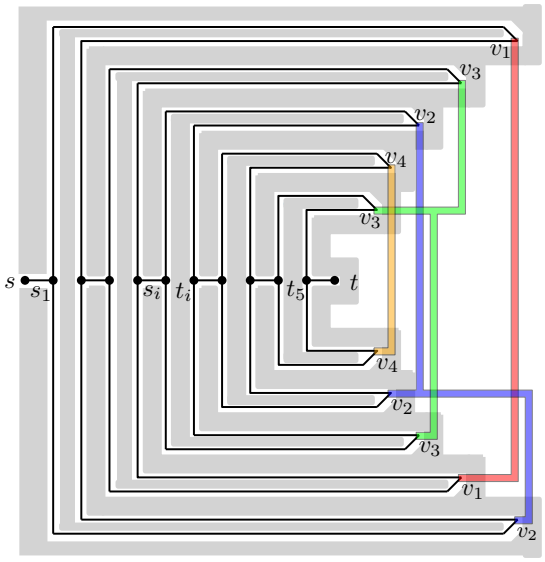


Figure 3.1: An example of the construction. The barrier region is shown in gray.

Lemma 19 *For any $1 \leq k \leq n$, there is a size k vertex cover for G iff there is an s-t path that intersects k obstacles.*

Proof: Suppose there is a size k vertex cover for G. Now for each edge (u, v), if u is in the vertex cover, we choose the pathlet corresponding to (u, v) that intersects the

obstacle corresponding to u. Otherwise, we choose the pathlet corresponding to (u, v) that intersects the obstacle corresponding to v. The s-t path formed by these chosen pathlets intersects only k obstacles. Similarly, suppose there is an s-t path Π that intersects only k obstacles. If for an edge (u, v), Π contains the pathlet that intersects the obstacle corresponding to u, we select the vertex u. Otherwise, we select the vertex v. All the selected vertices form a vertex cover of size k. ■

As Vertex Cover is hard to approximate within a factor of $2 - \epsilon$ for any constant $\epsilon > 0$, assuming the Unique Games conjecture [24], we get the following theorem.

Theorem 11 *Minimum constraint removal with rectilinear polygons is hard to approximate within a factor of* $2 - \epsilon$ *for any constant* $\epsilon > 0$*, assuming the Unique Games conjecture.*

It is easy to see that the same idea can easily be extended for convex polygons. Basically, one can connect the tips of any subset of pathlets using segments to form a convex polygon that does not intersect any other pathlets.

Lemma 20 *Minimum constraint removal with convex polygons is hard to approximate within a factor of* $2 - \epsilon$ *for any constant* $\epsilon > 0$*, assuming the Unique Games conjecture.*

3.3.2 APX-hardness for Axis Parallel Rectangles

We reduce a restricted version of vertex cover, which is referred to as Special-3VC, to our problem. Chan et al. [25] introduced this version for the sake of proving APX-hardness of several geometric optimization problems.

Definition 6 *In a* Special-3VC *instance, we are given a graph* $G = (V, E)$*, where* V *contains* $5m$ *vertices* $\{v_{ij} \mid 1 \leq i \leq m, 1 \leq j \leq 5\}$*.* E *contains* $4m + n$ *edges - 4m of type 1 and n of type 2, where* $2n = 3m$*. Type 1 edges are of the form* $\{(v_{ij}, v_{i,j+1}) \mid$

$1 \leq i \leq m, 1 \leq j \leq 4\}$. *Type 2 edges are of the form* $\{(v_{pq}, v_{xy}) \mid 1 \leq p < x \leq m$, *and* q, y *are odd numbers*$\}$ *such that any vertex* v_{ij} *with odd index* j *appears in exactly one such edge.*

As each vertex v_{ij} with odd index j contributes exactly once in the type 2 edges, the number of type 2 edges is $3m/2 = n$. Chan et al. [25] proved that Special-3VC is APX-hard. Now we describe our reduction. The reduction is similar to the reduction for rectilinear polygons. We will have one obstacle corresponding to each vertex. Moreover, we construct two pathlets corresponding to each edge (u, v) such that one pathlet intersects the obstacle corresponding to u and the other intersects the obstacle corresponding to v. However, due to the simpler structure of the obstacles, here it is more complicated to ensure that the pathlets intersect the correct obstacles. The construction of the instance of MCR is as follows.

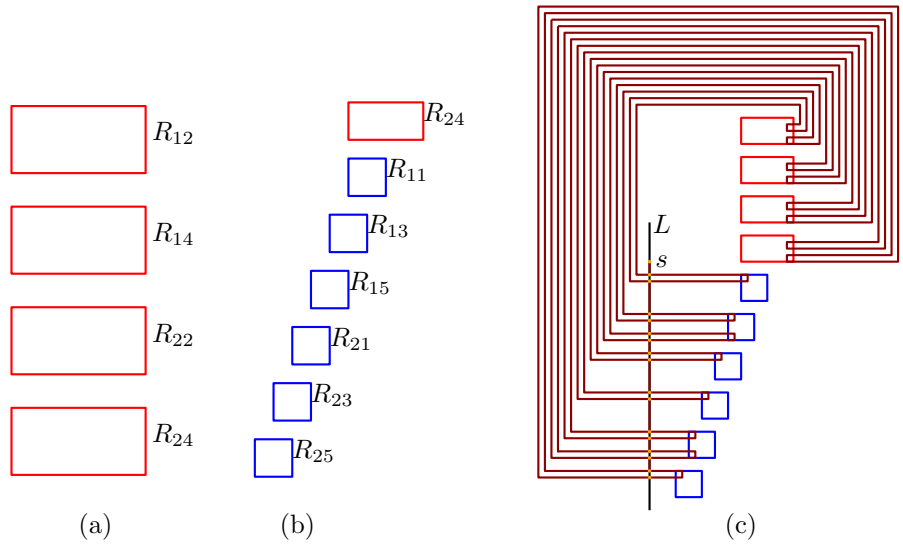


Figure 3.2: (a) The stack of the class 1 rectangles for $m = 2$. (b) The initial configuration of the class 2 rectangles (shown by squares) for $m = 2$. (c) Drawing of the pathlets for the class 1 edges.

We denote the rectangles corresponding to v_{ij} by R_{ij}. First we place the rectangles corresponding to the vertices in $\{v_{ij} \mid 1 \leq i \leq m$ and j is even$\}$ in a way so that they

form a stack like structure (see Figure 3.2(a)). Also the rectangles are placed from top to bottom in the lexicographic order of the indexes (i, j): R_{ab} is considered before R_{cd} if $a < c$, and R_{a2} is considered before R_{a4}. We refer to these rectangles as the class 1 rectangles. Thereafter we place the rectangles corresponding to the remaining vertices. All these rectangles are placed in lexicographic order of the indexes (i, j). The first one is placed below R_{m4} (the last rectangle of the stack) in a way so that its left side is aligned with the left side of R_{m4}. Thereafter every rectangle is placed below the already placed ones and a little aligned towards the left w.r.t. the previous one (see Figure 3.2(b)). We refer to these rectangles as the class 2 rectangles. We note that initially every class 2 rectangle is a square. Later each such rectangle might be expanded suitably towards right and below to ensure the correctness of the intersections with the pathlets.

Now let L be a vertical line such that all the rectangles are placed strictly to the right of it. All the endpoints of the pathlets we draw lie on L. Each pathlet is a curve consisting of rectilinear segments. The start (resp. end) points of the two pathlets corresponding to an edge are the same. We place s right above the topmost start point of the pathlets and connect s with this point by a vertical segment. Similarly, the point t is placed below the bottommost end point and joined with it by a vertical segment. At first we draw the pathlets for type 1 edges $\{(v_{ij}, v_{i,j+1}) \mid 1 \leq i \leq m, 1 \leq j \leq 4\}$ in the dictionary order of the indexes $(i, j, i, j + 1)$, i.e at first (v_{11}, v_{12}), then (v_{12}, v_{13}) and so on. The pairs of start and end points of the pathlets corresponding to these edges appear in the same order on L from top to bottom. For each type 1 edge $(v_{ij}, v_{i,j+1})$, let $s(i, j, j + 1)$ and $t(i, j, j + 1)$ be the respective start and end points of the pathlets. Note that for a v_{ij} with odd j, v_{ij} appears in two type 1 edges only if j is 3. Otherwise, it appears only once. Let P_{ij} be the horizontal projection (an interval) of R_{ij} on L. Then the start and endpoints of the pathlets of the type 1 edges with a vertex v_{ij} lie on P_{ij}. Now consider a type 1 edge $(v_{ij}, v_{i,j+1})$. Then either j or $j + 1$ is odd. WLOG let j is odd. We draw the two points

$s(i, j, j+1)$ and $t(i, j, j+1)$ on P_{ij} such that $s(i, j, j+1)$ lies above $t(i, j, j+1)$. One pathlet of $(v_{ij}, v_{i,j+1})$ lies on the right of L. It consists of three orthogonal segments and the only rectangle it intersects is R_{ij} (see Figure 3.2(c)). The other pathlet is also drawn in a way so that the only rectangle it intersects is $R_{i,j+1}$ (see Figure 3.2(c)). We repeat the process for all type 1 edges and each consecutive pairs of end and start points are joined with a vertical segment.

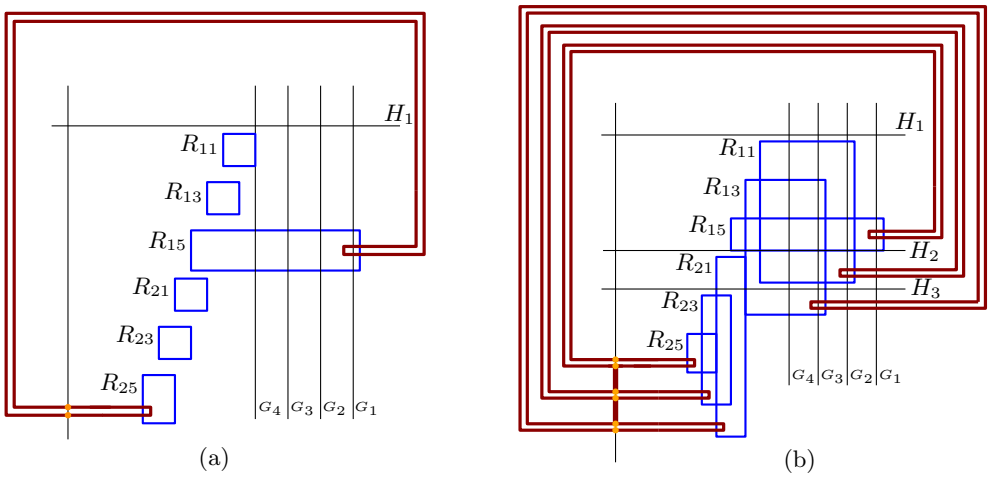


Figure 3.3: (a) Drawing of the pathlets for the edge (v_{pq}, v_{m5}) where $m = 2, p = 1, q = 5$. (b) Drawing of the pathlets for the type 2 edges $(v_{15}, v_{25}), (v_{11}, v_{23}), (v_{13}, v_{21})$ where $m = 2$.

Now we draw the pathlets corresponding to the type 2 edges $\{(v_{pq}, v_{xy}) \mid 1 \leq p < x \leq m,$ and q, y are odd numbers$\}$. Note that, there are n such edges in G. We process all these edges in the reverse lexicographic order of the indexes (x, y) of the vertices v_{xy}. Thus at first we consider the edge that contains v_{m5}, then the edge that contains v_{m3} (if not considered already), then the edge that contains v_{m1}, then the edge that contains $v_{m-1,5}$ (if not considered already), and so on. We take $n+1$ vertical lines $G_1, \ldots, G_{n+1}$ such that G_{n+1} intersects the right vertical side of R_{11}, G_n is on the right of G_{n+1}, G_{n-1} is on the right of G_n, and in general G_i is on the right of G_{i+1}. Also let G_i and G_{i+1} are unit distance apart for $1 \leq i \leq n$. In every iteration $1 \leq i \leq n$, we define a horizontal line H_i.

Denote by Q_i the region that lies below H_i and inside the strip defined by G_i and G_{i+1}. The drawing procedure is the following. Consider the first edge (v_{pq}, v_{m5}) corresponding to the vertex v_{m5}. Let H_1 be a horizontal line such that all the class 1 rectangles lie above it and all the class 2 rectangles lie below it. At first we expand R_{m5} sufficiently towards below such that one can place a pathlet with the following properties - the only rectangle it intersects is R_{m5}, it consists of two horizontal segments and one vertical segment, and its start and end points lie on L. Note that the expansion of R_{m5} do not create any new intersections with the existing pathlets. Thereafter R_{pq} is expanded sufficiently towards below and right to ensure that it has non-empty intersection with Q_1. Then the other pathlet can be drawn in a way so that it intersects the portion of R_{pq} that is in Q_1, and as Q_1 is empty the pathlet does not intersect any other rectangle (see Figure 3.3(a)). Now consider the i^{th} type 2 edge (v_{pq}, v_{xy}) in this order. Let all the edges before it in the ordering are already taken care of. It is easy to see that one can expand R_{xy} towards below for drawing a pathlet with the desired properties. Now to make sure that the other pathlet intersects only R_{pq}, set H_i to be a horizontal line such that the region Q_i, as defined above, is empty of previously drawn pathlets and expanded rectangles. Then we can expand R_{pq} towards below and right so that it has non-empty intersection with Q_i. As Q_i is empty one can draw the other pathlet as well with the desired properties (see Figure 3.3(b)).

Lastly, we draw the barrier region around the paths. As the pathlets are orthogonal and consisting of a polynomial number of segments in total, the barrier region can be simulated using a polynomial number of rectangles and thus the construction can be realized in polynomial time. From the construction, it is straightforward to see the following lemma.

Lemma 21 *For any $1 \leq k \leq |V|$, there is a size k vertex cover for G iff there is an s-t*

path that intersects k rectangles.

As Special-3VC is APX-hard we obtain the following theorem.

Theorem 12 *Minimum constraint removal with rectangles is APX-hard.*

3.4 Bibliographic Notes

The geometric minimum constraint removal problem has been studied under different names across multiple research communities, including sensor networks and robotics. In networks, the problem is called barrier resilience where obstacles represent coverage regions of sensors, and the network's resilience is measured by the minimum number of sensors whose removal creates a sensor-avoiding s–t path. Circular disks are a commonly used model for sensor coverage. When all disks have the same (unit) radius, a 2-approximation is known due to Chan and Kirkpatrick [26], who build and improve upon the earlier work of Bereg and Kirkpatrick [16]. However, even for this simple case, the complexity of minimum constraint removal is an unsolved open problem [26]. In [26, 5], constant factor approximations are proposed for restricted versions of arbitrary radii disks. However, when disks have arbitrary radii, no sub-linear approximation with provable guarantee is known. The problem has also been studied for other types of obstacles, mainly from the perspective of time complexity. The problem has been shown to be NP-complete for convex obstacles [1], for line segments [2], even in the bounded density case [3, 4], and for axis-parallel rectangles with aspect ratio close to one [5].

In robotics, the minimum constraint removal problem models the motion planning problem of multi-articulated robot [27, 1]. Suppose we have a physical environment containing a disjoint set of impenetrable obstacles in the plane, and a robot with two degrees of freedom. Then the configuration space approach to motion planning shrinks the

robot to a point while simultaneously expanding the obstacle by taking their Minkowski sum with the robot's geometry. The result is our minimum constraint removal problem: a set of two-dimensional intersecting obstacles that may have no feasible path for the robot, and so some obstacles need to be removed. Finally, the problem has also been studied through the lenses of parameterized complexity [4, 5], and exact and heuristic algorithms [27, 28].

Chapter 4

An $O(\log n)$ Approximation for MCR

In the previous chapter, we gave close to $O(\sqrt{n})$-approximation algorithm for MCR with polygonal obstacles and pseudodisks, where n was the number of vertices (in case of polygonal obstacles) or the number of pseudodisks. In this chapter, we work towards improving that bound. In fact, we will discuss an $O(\log |\mathcal{C}|)$-approximation[1] algorithm for the more general problem of computing a min-color path on a vertex-colored planar graph $G = (V, E, \mathcal{C})$ with *color connectivity*. That is, for any given color $C_i \in \mathcal{C}$, the set of vertices $V_i \subseteq V$ that contain C_i are connected in G. We will use the shorthand Planar-Conn-MCP to refer to this case. Recall that MCR is a special case of Planar-Conn-MCP over the *arrangement* graph of obstacles. Since MCR is NP-hard, it follows that Planar-Conn-MCP is also NP-hard.

Note that both these conditions (planarity and color connectivity) are crucial for existence of an $O(\log |\mathcal{C}|)$-approximation algorithm for Planar-Conn-MCP because without either of them, the problem is conditionally hard to approximate within a factor

[1] In the case of MCR, colors $\mathcal{C}$ correspond to the set of obstacles, so $|\mathcal{C}| \leq n$.

significantly better than $O(|\mathcal{C}|)$. (See Section 2.2, Theorem 1.)

We will now work towards designing an $O(\log |\mathcal{C}|)$-approximation algorithm for Planar-Conn-MCP. One of the key insights here is an alternative characterization in terms of *color separators* and finding a minimum size color set that *hits* all color separators. The remainder of this chapter is organized as follows. In Section 4.1, we introduce the notion of color separators and prove some useful properties. In Section 4.2 we use these ideas to obtain a linear program for Planar-Conn-MCP. Finally, in Section 4.4 we apply the well-known rounding technique of Leighton and Rao [29] to obtain our approximaion algorithm.

4.1 Color Separators

We begin by setting some basic definitions. Let $G(V, E, \mathcal{C})$ be a vertex-colored graph, and let s, t be two fixed vertices. For any set $\mathcal{C}' \subseteq \mathcal{C}$ of colors, we define its *host vertex-set* $V(\mathcal{C}') \subseteq V$ to be the set of vertices that contain a color in $\mathcal{C}'$. That is, $V(\mathcal{C}') = \{v \in V \mid \chi(v) \cap \mathcal{C}' \neq \emptyset\}$. We can now define color separators formally as follows.

Definition 7 (Color Separator) *A non-empty set of colors $S \subseteq \mathcal{C}$ is called an s-t* color separator*, if removing the host vertex-set $V(S)$ of S from the graph G separates the vertices s and t into distinct connected components.*

Moreover, we say that a vertex $v \in V$ is a *white vertex* if $\chi(v) = \emptyset$. That is, v does not contain any color. Recall that our goal is to find a min-color path from s to t, so we can assume that both s and t are white vertices, because any s-t path must include all colors from both s and t. From the definition of color seperators, we note the following.

Observation 2 *Let S be an s-t color separator of G, then its host vertex-set $V(S)$ contains no white vertices.*

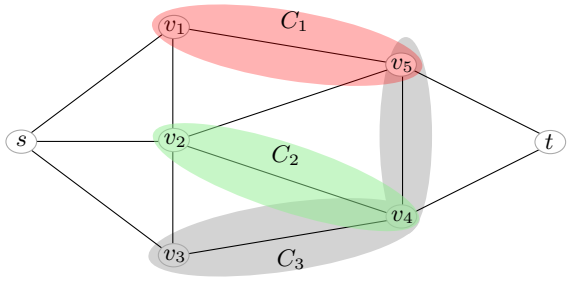


Figure 4.1: An instance of Planar-Conn-MCP with three colors $\mathcal{C} = \{C_1, C_2, C_3\}$. The colorsets $\{\{C_3\},\ \{C_1, C_2\},\ \{C_1, C_3\},\ \{C_1, C_2, C_3\}\}$ are all color separators. The color set $S = \{C_3\}$ is the minimum color separator. Note that removing its host vertex set $V(S) = \{v_3, v_4, v_5\}$ separates s from t.

Clearly, it is possible that a graph G does not contain any color separators. However, if that happens, we have the following useful property.

Lemma 22 *If G does not contain any color separator, then there exists an $s-t$ path in G containing only white vertices.*

Proof: Consider the trivial color set $S = \mathcal{C}$. Since G does not contain any color separator, we must have that s and t remain connected even when vertices of $V(S)$ are removed from G. Moreover, all vertices of $G - V(S)$ are white. Therefore, there must exist a white path from s to t in $G - V(S)$, which is also a path in G. ■

Suppose we define $\mathcal{S}$ to be the set of all color separators of G. If $\mathcal{S} = \emptyset$, Lemma 22 easily gives us a path that uses a minimum number (zero) of colors. However, it is likely that $\mathcal{S}$ contains a large number (possibly exponential) number of color separators. We can still have the following lemma.

Lemma 23 *If $\mathcal{C}^*$ is the set of colors such that $S \cap \mathcal{C}^* \neq \emptyset$ for all $S \in \mathcal{S}$ (colors $\mathcal{C}^*$ "hit" all color separators), then there exists a path π such that the colors used by it $\chi(\pi) \subseteq \mathcal{C}^*$.*

Proof: First, we remove colors of $\mathcal{C}^*$ from vertices of $G = (V, E, \mathcal{C})$ to obtain a modifed colored graph $G' = (V, E, \mathcal{C} \setminus \mathcal{C}^*)$. That is for all vertices $v \in V$, we assign $\chi(v) = \chi(v) \setminus \mathcal{C}^*$ in G'.

Next, we claim that G' does not contain a color separator. For the sake of contradiction, suppose G' contains a color separator S'. Then, removing the vertices $V(S')$ disconnects s from t. Clearly, S' is also a color separator in G such that $S' \cap \mathcal{C}^* = \emptyset$ which contradicts the choice of $\mathcal{C}^*$.

By Lemma 22, there must be a path π in the modified graph G' containing only white vertices. Therefore, the set of colors of π in the original graph $\chi(\pi) \subseteq \mathcal{C}^*$. ■

We will now formally define the *min-color hitting set* problem on a colored graph and in the next lemma show that it is equivalent to min-color path.

Definition 8 (Min-Color Hitting Set) *Given a vertex-colored graph $G = (V, E, \mathcal{C})$ and let $\mathcal{S}$ be the set of all color separators of G. In the* min-color hitting set *problem, we want to compute the smallest sized set of colors $\mathcal{C}^* \subseteq \mathcal{C}$ that hits every seperator in $\mathcal{S}$. That is, $S \cap \mathcal{C}^* \neq \emptyset$ for all $S \in \mathcal{S}$.*

Lemma 24 *Given a vertex-colored graph $G = (V, E, \mathcal{C})$, the problem of computing a* min-color path *and computing a* min-color hitting set *are equivalent.*

Proof: Let π^* be the min-color path and let $\mathcal{C}^*$ be the min-color hitting set of G. Moreover, let $\mathcal{S}$ be the set of all color separators of G.

For the forward direction, it is easy to see that the min-color path π^* corresponds to a set of colors that hit all separators in $\mathcal{S}$. In particular, consider the color set $\chi(\pi^*) = \bigcup_{v \in \pi} \chi(v)$. If $\chi(\pi^*)$ did not hit all color sperators in $\mathcal{S}$, then must have a color separator $S' \in \mathcal{S}$ such that its host vertex-set $V(S')$ is disjoint from π. Since π will remain an s–t path in $G - V(S')$, removing $V(S')$ from G does not disconnects s from t which contradicts that S' is a color separator. Therefore, we must have that $\chi(\pi^*)$ hits all color separator and $|\mathcal{C}^*| \leq \chi(\pi^*)$.

For the other direction, applying Lemma 23, it follows that $\chi(\pi^*) \leq |\mathcal{C}^*|$. Therefore,

we must have $\chi(\pi^*) = |\mathcal{C}^*|$, and a solution to min-color path corresponds to a solution to min-color hitting set and vice versa. ■

Since the number of all color separators in $\mathcal{S}$ can be arbitrarily large, translating a min-color path instance to min-color hitting set instance is not directly useful to obtain approximation algorithms. Instead, we use the equivalence of two problems to obtain an LP formulation for min-color path. Then we apply rounding techniques on this LP to obtain an $O(\log |\mathcal{C}|)$-approximation algorithm for Planar-Conn-MCP. We note that planarity and color connectivity are both crucial for our approximation algorithm.

4.2 An LP Formulation

We will now present the following linear program whose integral solutions are solutions to min-color hitting set on the graph G. For each color $C_i \in \mathcal{C}$, we associate a variable $0 \leq x_i \leq 1$ that indicates the whether or not the color C_i is included in the solution. We then have the following formulation which we will refer to as Hitting-LP .

$$\min \quad \sum_{c \in \mathcal{C}} x_i$$

such that:

$$\sum_{C_j \in S} x_j \geq 1 \qquad \text{for all color separators } S \in \mathcal{S} \tag{4.1}$$

The set of constraints ensure that every color separator in $\mathcal{S}$ contains at least one color that is included in the solution. Note that the number of constraints can be exponential.

In order to obtain an approximation algorithm using this LP, we first need to be able to solve it in polynomial time. Although the LP contains an exponential number of constraints, it may still be possible to solve it in polynomial time using ellipsoid method

provided the existence of a polynomial time *separation oracle*. In the following, we introduce the notion of a *min-color separator* which will serve as a separation oracle for the above LP.

Definition 9 *(Min-Color Separator) Let $G = (V, E, \mathcal{C})$ be a weighted colored graph such that each color $C_i \in \mathcal{C}$ has a non-negative weight w_i. We define a* min-color separator *of G to be a color separator $S \in \mathcal{S}$ that minimizes the weight $w(S) = \sum_{C_j \in S} w_i$.*

We can now have the following lemma.

Lemma 25 *If there exists a polynomial time algorithm to find a min-color separator of G, then* Hitting-LP *can be solved in polynomial time.*

Proof: Roughly speaking, we use the algorithm for min-color separator of G as a polynomial time separation oracle for ellipsoid method. Specifically, given a solution vector $\hat{x} = \langle x_1, x_2, \ldots, x_{|\mathcal{C}|} \rangle$, a separating oracle decides if $\hat{x}$ is feasible, and if it is infeasible produces finds a hyperplane separating $\hat{x}$ and the feasible region. Setting the weights of a color to its fractional value x_i, we can compute a min-color separator $S^* \in \mathcal{S}$ in polynomial time. If $w(S^*) \geq 1$, then $\hat{x}$ is feasible. Otherwise, we have found a violating constraint S^* which corresponds to the separating hyperplane. ■

Indeed, as shown in Section 4.5, computing a min-color separator is NP-hard in general. However, the problem is polynomial time solvable on vertex-colored graphs that are planar and have color-connectivity (instances of Planar-Conn-MCP). The algorithm is discussed in Section 4.5.1. Here, we state the resulting lemma.

Lemma 26 *The* Hitting-LP *can be solved in polynomial time on planar graphs with color-connectivity.*

Using the above lemma and the equivalence of min-color path and min-color hitting set, it follows that one can obtain a *fractional* solution $\hat{x} = \langle x_1, x_s, \ldots, x_{|\mathcal{C}|} \rangle$ for Planar-Conn-MCP in polynomial time. That is, if OPT is the number of colors used by the

min-color path, then for each color $C_i \in \mathcal{C}$, we can compute a fractional value x_i such that $\sum x_i \leq OPT$. We now need to *round* the fractional solution vector $\hat{x}$ to obtain an integral solution $\hat{y}$ such that $\sum y_i \leq O(\log |\mathcal{C}|) \cdot OPT$. Towards that end, we first establish some useful properties for color separators. Unless otherwise stated, we assume that the graph G is color-connected and planar. We discuss the details of the rounding scheme in Section 4.4.

4.3 Structural Properties of Color Separators

In this section, we will discuss some structural properties of color separators on a color-connected planar graph $G = (V, E, \mathcal{C})$. We begin by fixing an embedding of G and let $G^* = (V^*, E^*)$ be its dual graph. A *separating cycle* of G^* is a cycle that separates s from t. We then have the following lemma.

Lemma 27 *Every color separator S of graph G corresponds to a non-empty family of separating cycles $\mathcal{F}(S)$ of the dual graph G^*.*

Proof: Let $E(S) \subseteq E$ be the set of edges adjacent to vertices in $V(S)$. Consider any set $E_\gamma \subseteq E(S)$ such that removing E_γ from G separates s from t. That is, edges E_γ are *cut edges*. Given a set of cut edges, it is not hard to obtain a separating cycle γ in the dual graph : draw a simple closed curve enclosing one of s or t and only intersecting the cut edges.

Repeating this for all possible E_γ gives a family $\mathcal{F}(S)$ of separating cycles. Note that $\mathcal{F}(S)$ is non-empty because removing $V(S)$ separates s from t, so the set $E_\gamma = E(S)$ is a trivial cut edge set. ■

Therefore, for each color separator S, we can associate a separating cycle $\gamma \in \mathcal{F}(S)$ in the dual graph. Next, we assign colors to the vertices of the dual graph so that the colors

on vertices of γ correspond to the colors in S.

Assigning Colors to the Dual Graph Let $v^* \in V^*$ be a vertex in the dual graph G^* and let f be its corresponding face in G. We assign to v^* the union of all colors on the boundary vertices of f. That is, if ∂f denotes the set of boundary vertices of f, we assign $\chi(v*) = \bigcup_{v \in \partial f} \chi(v)$. We now note the following.

Lemma 28 *The colored dual graph G^* is also color-connected.*

Proof: Let v_i^*, v_j^* be two vertices in the dual corresponding to faces f_i, f_j of G. Consider a color $C \in \chi(v_i^*) \cap \chi(v_j^*)$. By color-connectivity of G, there must be a path $\pi = v_1 \to v_2 \to \cdots \to v_r$ in G from a vertex v_1 on the boundary of f_i to a vertex v_r on boundary of f_j. We want to find a color-connected path π^* from v_i^* to v_j^* in the dual.

We do an induction on number of vertices r on the path π. If v_2 also lies on face f_i, we can drop v_1 from π and we are done by induction. If v_2 lies on some other face f_k, we consider the clockwise order of faces around v_1- both f_i, f_k are faces in this ordering. Moreover, all these faces will contain the color C. Traversing faces in this order, starting from v_i^* we can reach v_k^* (dual vertex of f_k) by a path in dual such that every vertex on this path contains color C. Now, we can consider the subpath of π from v_2 to v_r and will again be done by induction. ■

Now that we have added colors to the dual graph G^*, we want to assign an ordering of colors of the separator S on the cycle γ. We do this by computing a *color-mapping* $\mathcal{M}$ that assigns to every vertex $v^* \in \gamma$, a color from $S \cap \chi(v^*)$. Moreover, we ensure that this color mapping 'respects' the order in which γ intersects the cut edges E_γ. (Recall from proof of Lemma 27, that every separating cycle $\gamma \in \mathcal{F}(S)$ corresponds to a set of cut edges $E_\gamma \subseteq E(S)$.) Observe that edges in the cycle γ are basically the dual of E_γ.

Computing a Color-Mapping $\mathcal{M}$ Assume that the vertices on γ are arranged in clockwise order: $v_1^* \to v_2^* \to \ldots v_r^* \to v_1^*$. With each vertex $v_i^* \in \gamma$, we now associate a valid-colorset $\mathcal{C}(v_i^*)$ as follows.

Definition 10 (Valid-Colorsets) *Let v_{i-1}^* be the predecessor of v_i^* on γ, and let $e^* = (v_{i-1}^*, v_i^*)$ be the corresponding edge in G^*. Let $e = (u, u') \in E_\gamma$ be the primal edge corresponding to e^*. Then, the valid-colorset for v_i^* is given by $\mathcal{C}(v_i^*) = (\chi(u) \cup \chi(u')) \cap S$.*

Lemma 29 *For every $v_i^* \in \gamma$, the valid-colorset $\mathcal{C}(v_i^*)$ is non-empty.*

Proof: Recall that $E_\gamma \subseteq E(S)$, where $E(S)$ is the set of edges adjacent to host-vertex set $V(S)$ of color separator S. Therefore for every edge $e = (u, u') \in E_\gamma$, at least one of u or w lies in $V(S)$. Therefore, the color set $\chi(u) \cup \chi(w)$ contains at least one color from S. ■

The *color-mapping* $\mathcal{M}$ is simply a mapping from a vertex $v_i^* \in \gamma$ to any color in its valid-colorset $\mathcal{C}(v_i^*)$. Similarly, we can extend the mapping $\mathcal{M}$ to all vertices $v_i^* \in \gamma$. This gives us a cyclic sequence of colors $\mathcal{M}(v_1^*) \to \mathcal{M}(v_2^*) \ \ldots \ \rightsquigarrow \ \ldots \ \mathcal{M}(v_r^*) \to \mathcal{M}(v_1^*)$ which we will refer to as *color-cycle* of γ and denote it by $\mathcal{M}(\gamma)$.

Intuitively, $\mathcal{M}$ simply maps a separating cycle $\gamma \in \mathcal{F}(S)$ to a color-cycle $\mathcal{M}(\gamma)$ by *selecting* one color belonging to S from every vertex on γ. Note that given γ and its color-cycle we can easily obtain the corresponding mapping and vice-versa. The following lemma is easy to verify.

Lemma 30 *Let v_{i-1}^*, v_i^* be two consecutive vertices on the clockwise ordering of separating cycle γ. Then the color $\mathcal{M}(v_i^*)$ lies in $\chi(v_{i-1}^*)$.*

Proof: This follows easily from how we define valid-colorset $\mathcal{C}(v_i^*)$. Indeed, the vertices u, u' in Definition 10 lie on the face f_{i-1} corresponding to the dual vertex v_{i-1}^*. Therefore, we must have $\mathcal{C}(v_i^*) \subseteq \chi(v_{i-1}^*)$. Since $\mathcal{M}(v_i^*) \in \mathcal{C}(v_i^*)$, the lemma follows. ■

Well-behaved Separating Cycles We know from Lemma 27 that there can be multiple separating cycles $\mathcal{F}(S)$ for any given color separator S. We now establish the notion of a *well-behaved* separating cycle and then show that a well-behaved separating cycle always exists.

Definition 11 (Well-behaved Separating Cycles) *We say that a color-cycle Z is* well-behaved *if all occurrences of any given color C in Z are consecutive. We say that γ is a* well-behaved separating cycle *if there exists a mapping $\mathcal{M}$ such that the color-cycle $\mathcal{M}(\gamma)$ is well-behaved.*

As an example, the color-cycle $Z = C_1 \to C_2 \to C_2 \to C_1$ is well-behaved, but $Z = C_1 \to C_2 \to C_3 \to C_2 \to C_1$ is not well-behaved.

We now show that such a cycle always exists. Roughly speaking, the overall idea is to start with an arbitrary separating cycle $\gamma \in \mathcal{F}(S)$ and modify it so that it stays a separating cycle and also becomes well-behaved.

Let v_i^*, v_j^* be any two vertices on γ. Suppose we split the cycle γ into two disjoint paths π_1, π_2 from v_i^* to v_j^*. That is, $\gamma = \pi_1 \oplus \pi_2$, where $\oplus$ denotes the operation of concatenating two paths at their common endpoints. We can then obtain the following lemma.

Lemma 31 *Let u^* and v^* be two vertices on γ such that both contain a given color C. Moreover, let π be a path from u^* to v^* in G^* that is* disjoint *from γ and such that all vertices on π also contain color C. Then, exactly one of $\gamma_1 = \pi_1 \oplus \pi$ and $\gamma_2 = \pi_2 \oplus \pi$ is a separating cycle.*

Proof:

Note that since π is disjoint from γ, both γ_1 and γ_2 are simple cycles. Let R be the region enclosed by γ, we have the following two cases. We assume that source s lies inside R and destination t lies outside R.

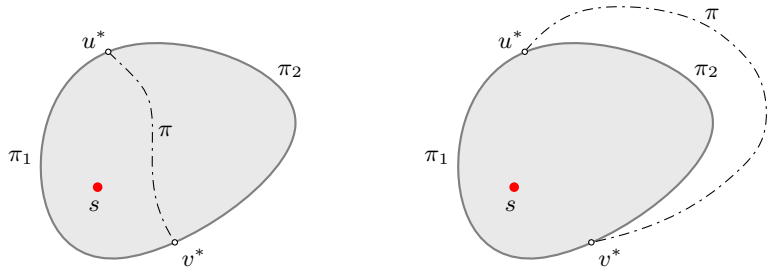


Figure 4.2: Two cases from proof of Lemma 31. Separating cycle γ (shown in bold) is split into paths π_1 and π_2. (a) Path π is disjoint from γ and lies inside (b) Path π is disjoint from γ and lies outside.

1. *Path π lies inside R*: In this case, π splits the region R into two disjoint sub-regions defined by γ_1 and γ_2. We set γ' to whichever of γ_1 or γ_2 contains the source s. (See also Figure 4.3(a).)

2. *Path π lies outside R*: In this case, one of the two cycles γ_1 and γ_2 completely encloses the region R and the other is disjoint from R. Without loss of generality assume that γ_1 encloses R and γ_2 is disjoint from R. If t lies outside γ_1, we set γ' to γ_1 as our separating cycle. If t lies inside γ_1, we set γ' to γ_2. (See also Figure 4.3(b).)

In both these cases, we were able to find another separating cycle γ', which is basically π concatenated with either π_1 or π_2. ■

Lemma 32 *Every color separator S has a separating cycle that is well-behaved.*

Proof: Let $\gamma \in \mathcal{F}(S)$ be any separating cycle. Let $\mathcal{M}$ be a trivial mapping that simply assigns to each vertex $v^* \in \gamma$ any arbitrary color from its valid-colorset $\mathcal{C}(v_i^*)$. (From Lemma 29, at least one such color exists). Now, if the sequence $\mathcal{M}(\gamma)$ is well-behaved, we are done. If not, we will iteratively transform γ to obtain another separating cycle that is well-behaved.

Let u^* and v^* be two non-consecutive vertices on γ that *violate* the well-behaved property. That is, both u^* and v^* are mapped to a color C but the intermediate vertices

are mapped to a different color. We will find another separating cycle γ' that eliminates this violation (without adding any extra violations). Therefore, γ' has at least one less violation than γ. Applying this process repeatedly gives us a well behaved separating cycle (one with zero violations).

Since both u^* and v^* contain the color C, using color connectivity of G^* (Lemma 28), there must be a simple path π from u^* to v^* such that all intermediate vertices on this path also contain the color C. If π was disjoint from γ, then we can simply apply Lemma 31 to obtain a separating cycle in which u^* and v^* are connected by path π. Setting $\mathcal{M}(v) = C$ for all $v \in \pi$ gives us a separating cycle that eliminates the $u^* - v^*$ violation, and clearly does not add any extra violations.

However, it is possible that π intersects γ multiple times, which makes things a little more complicated. We claim that even in this case, there exists a separating cycle that eliminates the $u^* - v^*$ violation. The proof of this is by induction on X, which is the number of times π intersects γ. We just discussed the base case of $X = 0$. That is, when π is disjoint from γ and found a desired separating cycle.

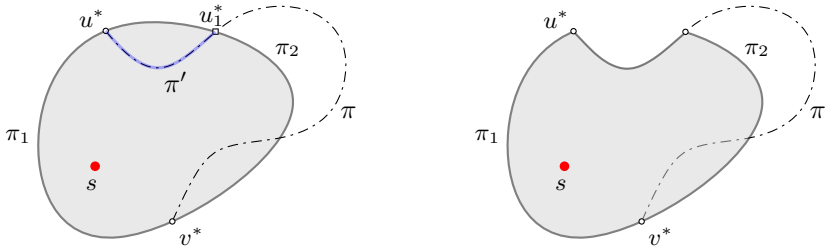


Figure 4.3: The case when color-connecting $u^* - v^*$ path π crosses γ multiple times. The modified separating cycle γ' to the right contains a $u^* - v^*$ path with one fewer intersection.

For the inductive step, find the vertex u_1^* closest to u^* where π crosses γ. (See also Figure 4.3.) The sub-path $\pi' : u^* \rightsquigarrow u_1^*$ along π is disjoint from γ. Applying Lemma 31 again with π' gives us a separating cycle γ'. Note that π' is now a part of γ'. Moreover, we assign $\mathcal{M}(v) = C$ for all $v \in \pi'$. One can verify that γ' either eliminates the $u^* - v^*$

violation, in which case we are done. Otherwise, there exists a $u^* - v^*$ path containing color C that intersect γ' at most $X - 1$ times. In the latter case, we are done by induction.

■

In the rest of our discussion, whenever we say a separating cycle γ is well behaved, we assume that we are also given a well-behaved color-cycle $\mathcal{M}(\gamma)$. Indeed one compute $\mathcal{M}(\gamma)$ using Lemma 32. We conclude this section with one last property of separating cycle that will be used later.

Lemma 33 *Let γ be a simple closed curve in the plane and let π_{st} be a simple path between two points s, t in the plane. Then γ* separates *s and t if and only if the number of times π_{st} intersects γ is odd.*

Proof: By Jordan curve theorem, γ divides the plane into two distinct regions *exterior* and *interior* of γ. Let R be the region interior of γ. Observe that γ separates s from t if one of them lies inside and the other lies outside. We show that this happens if and only if π intersects γ odd number of times.

($\Rightarrow$) Without loss of generality, assume s lies inside R and t lies outside. Then we follow the path π_{st}. If π_{st} *exits* R but does not *enter* R, the number of intersection is one. Assume that number of intersections is at least two, and consider the first two intersections. Since the first intersection is exit, the second must be entry. We can discard these intersections (two in total) and repeat the process. Eventually, all but the last intersection can be paired as *exit* $\rightsquigarrow$ *entry*. So the total number of intersections is odd.

($\Leftarrow$) We prove the contrapositive. That is if s, t are both outside or inside R, then the number of intersections is even. To see this, we follow the path π_{st} again, starting from s. The first intersection must be either an *entry* or an *exit*. In either case, we consider the point x immediately after the intersection which must lie inside R if both s, t were outside or it must lie outside R if both s, t were inside. Now x lies in the region opposite

of t, so from the proof for forward direction, the number of intersections from x to t is odd. Therefore, the total number of intersections from s to t must be even. ■

Combining Lemma 32 and 33, we have the following.

Lemma 34 *Let $G = (V, E, \mathcal{C})$ be a color-connected planar graph and $G^* = (V^*, E^*, \mathcal{C})$ be its* colored dual *graph. Moreover, let π_{st} be an arbitrary s–t path in G. Then any color separator S corresponds to a* well-behaved *cycle in G^* that crosses π_{st} an odd number of times.*

4.4 An $O(\log |\mathcal{C}|)$-Approximation Algorithm

In the previous sections, we introduced the notion of color separators, established some useful properties, and designed a linear program Hitting-LP for Planar-Connected-MCP and claimed that it can be solved in polynomial time (Lemma 26). Recall that a fractional solution of Hitting-LP corresponds to a solution vector $\hat{x} = \langle x_1, x_2, \ldots, x_{|\mathcal{C}|} \rangle$ such that $\sum_i x_i \leq OPT$ where OPT is the number of colors used by a min-color path. Note that OPT is also the objective function value for an integral solution of Hitting-LP . In this section, our goal is to round $\hat{x}$ to compute an integer solution vector $\hat{y}$ such that $\sum_i y_i \leq O(\log |\mathcal{C}|) \cdot OPT$.

Our rounding scheme is based on the well-known *small diameter graph decomposition* technique due to Leighton and Rao [29]. We will need a *node-weighted* version of this decomposition where distance values are on nodes and distance between two nodes is defined as the sum of the distance values on intermediate and destination nodes. We state their main theorem relevant to our algorithm.

Theorem 13 (Lemma 13, [29]) *Suppose we are given a graph $\mathcal{G}$ such each vertex $v \in V(\mathcal{G})$ has a nonegative cost $c(v)$ and a distance $d(v)$ associated with it. Moreover, let*

$W = \sum_{v \in V(\mathcal{G})} c(v) \cdot d(v)$ be the total weight of the distance function d. Then there exists a set $X \subseteq V(\mathcal{G})$ of vertices such that radius of each component in $\mathcal{G} \setminus X$ is at most δ and the cost of $c(X) = \sum_{v \in X} c(v)$ is $O(W \log |V(\mathcal{G})|/\delta)$

Intuitively, it may be convenient to argue about the diameter of a component by fixing a vertex v_c in the component and drawing a ball of some radius δ around it – *all vertices that are within a distance δ (using $d(v)$ values) from v_c lie in that component.* Clearly, diameter of the component is at most 2δ.

Applying Leighton-Rao decomposition In order to apply Theorem 13 to round the fractional solution $\hat{x}$, we first need to construct a graph using x_i as distance values, where we can reason about diameter of components and its connection to hitting all color separators of a colored graph $G = (V, E, \mathcal{C})$. To that end, we define the following *color-incidence* graph $\mathcal{I}_G$.

Definition 12 (Color-Incidence Graph) *Let $\mathcal{I}_G$ be a graph whose vertices $v_i \in V(\mathcal{I}_G)$ correspond to color $C_i \in \mathcal{C}$ and we add an edge (v_i, v_j) to $\mathcal{I}_G$ if the corresponding colors C_i and C_j occur on some vertex of G^*, the colored dual graph of G.*

We are now all set describe our approximation algorithm (See Algorithm 4). Recall that although Algorithm 4 returns a set of colors that hits all color separators of G, by equivalence of min-color path and min-color hitting set (Lemma 24), the same color set is also a solution for Planar-Conn-MCP.

The approximation bound follows from the following simple lemma.

Lemma 35 *If OPT is the optimal number of colors, the number of colors $|\mathcal{C}^*|$ returned by Algorithm 4 is $O(\log |\mathcal{C}|) \cdot OPT$*

Proof: Rounding up variables with $x_j \geq 1/2$ in Step 2 of the algorithm only increases the cost by a factor of 2. Moreover, in Step 5 of Algorithm 4 where we apply Theorem 13,

Algorithm 4 Approximate Planar-Connected-MCP

Input: A vertex-colored planar graph $G = (V, E, \mathcal{C})$ with color connectivity.
Output: A set of colors $\mathcal{C}^*$ that hits all color separators of G.

1. Using Lemma 26, solve Hitting-LP in polynomial time. Let $\hat{x} = \langle x_1, x_2, \ldots, x_{|\mathcal{C}|} \rangle$ be the fractional solution vector.
2. Include all colors C_j to the solution $\mathcal{C}^*$ such that $x_j \geq 1/2$.
3. Build the *color-incidence* graph $\mathcal{I}_G$ over the remaining colors $\mathcal{C} \setminus \mathcal{C}^*$.
4. For each color C_i, assign its fractional value x_i to the corresponding vertex v_i in $\mathcal{I}_G$. That is, $d(v_i) = x_i$. Moreover, set $c(v_i) = 1$.
5. Apply Theorem 13 on the node-weighted graph $\mathcal{G}_I$ with radius $\delta = 1/2 - \epsilon$. Let X be the set of cut vertices obtained from the theorem.
6. Add the set of colors corresponding to all nodes of X to $\mathcal{C}^*$. Return $\mathcal{C}^*$.

we have :

$$W = \sum_{v \in V(\mathcal{G})} c(v) \cdot d(v) = \sum x_i \leq OPT$$

$$|\mathcal{C}^*| = c(X) = \sum_{v \in X} c(v) = O(W \log |\mathcal{C}|)$$

Therefore, we have $|\mathcal{C}^*| = O(\log |\mathcal{C}|) \cdot OPT$ as claimed. ■

In the next two lemmas, we show that Algorithm 4 indeed computes a set of colors $\mathcal{C}^*$ that hits all color separators of G. Since we removed all colors with $x_i \geq 1/2$ (Step 2), we can assume that every color separator S contains at least three vertices. Because if not, the sum of x_i values for colors on S will be less than 1, contradicting the constraint corresponding to S in Hitting-LP . We note the following.

Lemma 36 *Every color separator S corresponds to a cycle in the color-incidence graph $\mathcal{I}_G$.*

Proof: From Lemma 32, we know that every color separator corresponds to a

separating cycle γ in the dual graph and a mapping $\mathcal{M}$ such that the color-cycle $\mathcal{M}(\gamma)$ is well-behaved. Consider any two consecutive vertices $v^*_{i-1}, v^*_i \in \gamma$ that are mapped to different colors. That is $C_j = \mathcal{M}(v^*_{i-1}), C_k = \mathcal{M}(v^*_i)$ and $C_j \neq C_k$. By Lemma 30, we know that the face f_{i-1} corresponding to the dual vertex v^*_{i-1} contains both colors C_j and C_k. Therefore, there must be an edge corresponding to C_j and C_k in $\mathcal{I}_G$.

Since $\mathcal{M}(\gamma)$ is well-behaved, all occurrence of a color are consecutive, which corresponds to staying at the same vertex in $\mathcal{G}_I$. Since $\mathcal{M}(\gamma)$ is a color-cycle, S corresponds to a cycle in $\mathcal{I}_G$. ■

Lemma 37 *The set of colors $\mathcal{C}^*$ returned by Algorithm 4 hits all color separators of G.*

Proof: We begin by noting that every color separator that contains a color C_i with fractional value $x_i \geq 1/2$ is already hit by the set $\mathcal{C}^*$ (Step 2 of the algorithm). Therefore, we can restrict our attention to color separators S such that for all $C_i \in S$ $x_i \leq 2$. Indeed every such separator corresponds to a cycle in $\mathcal{I}_G$ (Lemma 36). So if we can show that $\mathcal{C}^*$ hits all cycles in $\mathcal{I}_G$, we are done.

Let $S \in \mathcal{S}$ be an arbitrary color separator of G, and let $Z(S)$ be the set of vertices in $\mathcal{I}_G$ corresponding to the color set S. It suffices to show that $Z(S)$ is *not contained* in a single component of $\mathcal{I}_G \setminus Z(\mathcal{C}^*)$. This holds because if S spanned at least two components, we will have $\mathcal{C}^* \cap S \neq \emptyset$, and therefore $\mathcal{C}^*$ will hit all color separators S.

The proof is by contradiction. Suppose there exists a color separator S whose vertex set $Z(S)$ is contained in a single component κ of $\mathcal{I}_G$. Since $\delta = 1/2 - \epsilon$, we know from Theorem 13 that diameter of κ is strictly less than 1. Consider the shortest path tree $\mathcal{T}$ of the component κ rooted at some vertex, say u_i which corresponds to color C_i. We will now try to compute the *image* $\mathcal{T}^*$ of $\mathcal{T}$ in the dual graph G^*

1. Observe that each color C_i is connected and therefore can be drawn as a simple closed curve β_i surrounding all vertices of G^* on which the color occurs. Let r_i be a

representative point for color C_i anywhere inside the curve β_i.

2. We compute the image $\mathcal{T}^*$ as follows.

   (a) The vertex set of $\mathcal{T}^*$ consists of representative vertices r_i and dual vertices v_i^*.

   (b) Let V_{ij}^* be the set of all vertices of the dual graph that contain both colors C_i and C_j. For every edge $(u_i, u_j) \in \mathcal{T}$, add edges to $\mathcal{T}^*$ connecting r_i to all vertices in V_{ij}^* by a path that stays inside β_i. Similarly, add edges connecting r_j to V_{ij}^* by path that lies inside β_j.

Observe that since representative vertices are points in the plane and the edges added to $\mathcal{T}^*$ correspond to paths in the plane, the embedding of G^* can be extended to obtain an embedding of $\mathcal{T}^*$. However, it is possible that the edges of $\mathcal{T}^*$ intersect each other. Suppose we add dummy crossover vertices at the intersection points of edges of $\mathcal{T}^*$ to make it planar. We now have two cases.

1. *s and t lies in different faces of $\mathcal{T}^*$.* In this case we would have found a separating cycle γ that only uses colors from the component κ and has weight $w(\gamma)$ strictly less than 1 (because diameter of κ is less than 1). This contradicts the constraint corresponding to the colorset of γ in the Hitting-LP .

2. *s and t lie on the same face of $\mathcal{T}^*$.* This case is a little more involved. We begin by drawing a path π_{st} disjoint from $\mathcal{T}^*$. Let γ correspond to a cycle R in $\mathcal{I}_G$ that is contained in component κ. Edges of R are of two types: those that belong to $\mathcal{T}$ (*tree edges*) and those which do not belong to $\mathcal{T}$ (*non-tree edges*). The tree-edges of R correspond to edges in $\mathcal{T}^*$ which are disjoint from π_{st}. Recall that from Lemma 33, we know that any separating cycle γ must cross π_{st} an odd number of times. Since tree edges cross π_{st} and even number (zero) of times, there must exist a non-tree edge that crosses π_{st} an odd number of times.

Using this as edge as a shortcut to the path only using tree edges gives a separating cycle that has weight strictly less than 1, which is a contradiction.

■

4.5 Computing a Min-Color Separator

In this section, we show how to compute a color separator of minimum cardinality (or minimum weight in the case when colors are weighted) in polynomial time. As discussed before, if such an algorithm exists, we can obtain a fraction solution for HITTING-LP in polynomial time.

However, as shown in the following lemma, the problem of computing a min-color separator turns out to be NP-hard on general colored graphs. Fortunately, if the graph is color-connected and planar, we show that the problem is polytime solvable (Section 4.5.1). Recall that this is all we need for our approximation algorithm for PLANAR-CONNECTED-MCP (Algorithm 4).

Lemma 38 *Computing a min-color separator is NP-hard on general colored graphs.*

Proof: We have a simple reduction from the minimum hitting-set problem. In the hitting-set problem, given a collection $\mathcal{S}$ of sets and a universe U, we want to decide if there exists a set $X \subseteq U$, such that $|X| \leq k$ and $X \cap S_i \neq \emptyset$ for all $S_i \in \mathcal{S}$.

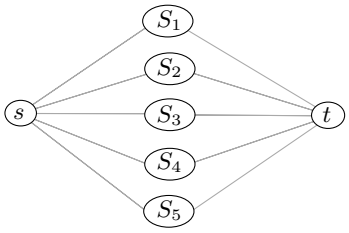


Figure 4.4: Reduction from Hitting Set

We construct a colored graph $G = (V, E, \mathcal{C})$ as follows. Create a node v_i for each set S_i and assign $\chi(v_i) = S_i$. Connect both s, t to all v_i. We show that there exists a color-separator of size k iff there exists a hitting set of size k. The colorset $\mathcal{C} = U$.

For the forward direction, observe that any color separator $X \subseteq \mathcal{C}$ with cardinality at most k, must have the host-vertex set $V(S) = V \setminus \{s, t\}$, and by definition of the color separator we can conclude that X is a hitting set for $\mathcal{S}$.

For the other direction, suppose we are given a hitting set X of cardinality k. Then, it is easy to see that the set of colors $X \subseteq U$ is a color separator. Observe that $V(X) = V \setminus \{s, t\}$, because for every $v_i \in V \setminus \{s, t\}$, we have $\chi(v_i) \cap X \neq \emptyset$. This holds because $\chi(v_i) = S_i$ and X is a hitting set. ■

We will now describe a polynomial time exact algorithm for Color-Connected planar graphs.

4.5.1 Exact Algorithm for Color-Connected Planar Graphs

Let $G = (V, E, \mathcal{C})$ be a color-connected planar graph such that each color $C_i \in \mathcal{C}$ has weight $w(C_i)$. Our goal is to compute a min-color separator. Recall from Definition 9, a min-color separator is a color separator S that minimizes $w(S) = \sum_{C_j \in S} w(C_j)$.

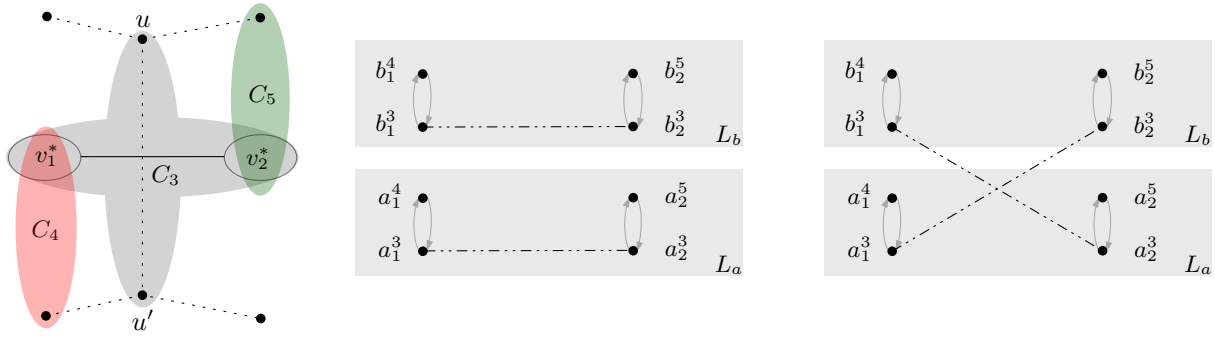
The key to an algorithm for min-color separator is Lemma 34, which states that every color separator S corresponds to a well-behaved cycle in the colored dual graph $G^* = (V^*, E^*, \mathcal{C})$ that crosses an arbitrary s-t path π_{st} an odd number of times. Recall that the notion of well-behaved means that all occurrences of a given color on the cycle are consecutive. This lets us formulate the problem of finding a min-color separator as a shortest path problem in an auxiliary *layered graph* H.

Constructing the Auxiliary Graph H

The auxiliary graph H consists of two layers: L_a and L_b with an identical set of vertices and two types of edges: *intra-layer* edges that go within the layer and *inter-layer* edges that go between layers. The graph H will be *edge weighted*. We first add vertices and edges to H and later assign weights to its edges.

Adding vertices to H For every dual vertex $v_i^* \in V^*$, we create $r = |\chi(v_i^*)|$ copies in L_a, one corresponding to each color in $\chi(v^*)$. More precisely, we add a vertex a_i^j to L_a for each pair (i, j) where $v_i^* \in V^*$ and $C_j \in \chi(v_i^*)$. We make another copy of all vertices in L_a and add them to L_b. We will refer to copy of vertex a_i^j as b_i^j in layer L_b.

Recall that G^* is a planar graph. So intuitively, we can think of following visualization of H in three dimensions. Let L_a be the bottom layer, L_1 to be the top layer, and stack all copies a_i^j, b_i^j of v_i^* one above another, such that all a_i^j copies come first followed by b_i^j. (See also Figure 4.5)



(a) An edge of G^* (b) edge does not crosses π_{st} (c) edge crosses π_{st}

Figure 4.5: An example of layered graph construction with weight of all colors as one. An edge (v_1^*, v_2^*) of colored dual graph G^* is shown in (a) and the two cases for its corresponding set of edges in H is shown in (b) and (c). The dash-dotted edges in the picture are *free edges* and have weight zero. All other edges have weight one.

Adding edges to H We add two groups of edges to H. The first group will be called *clique edges* and are added as follows. Let A_i be the set of all copies of vertex v_i^* in layer L_a. Add edges to H such that the vertex set A_i is a clique. Similarly, let B_i be the set of all copies of vertex v_i^* in layer L_b, add edges to H so that B_i is a clique. Repeat for all v_i^* Note that clique edges are intra-layer edges.

The second group of edges will be called *free edges* and are added as follows. For each edge $e^* = (v_x^*, v_y^*)$ of the dual graph G^*, we add a set of edges E_{xy} as follows depending on whether or not e^* *crosses* path π_{st}. Let $e = (u, u')$ be the primal edge corresponding to e^*. A dual edge e^* crosses a primal path π_{st} if the primal edge e lies on the path π_{st}. Suppose we define $\mathcal{C}_e = \chi(u) \cup \chi(u')$ to be the colorset associated with the primal edge e. We have the following two cases.

1. *Edge e^* does not cross π_{st}*: For every $C_j \in \mathcal{C}_e$, add edges (a_x^j, a_y^j) and (b_x^j, b_y^j). Note that all edges added in this case are also *intra-layer* edges. (See Figure 4.5(b))

2. *Edge e^* crosses π_{st}*: For every $C_j \in \mathcal{C}_e$, add edges (a_x^j, b_y^j) and (a_y^j, b_x^j). Note that all edges added in this case are *inter-layer* edges. (See Figure 4.5(c))

Assigning weights All *free edges* are assigned a weight of zero. We now assign weight to *clique edges*. Recall that *clique edge* are of the form (p_i^k, p_i^ℓ) where $p \in \{a, b\}$ and the superscript $\{k, \ell\}$ denote the indices of the color $C_k, C_\ell \in \mathcal{C}$. First, we make these edges directed by adding two directed edges $(p_i^k \to p_i^\ell)$ and $(p_i^k \leftarrow p_i^\ell)$. We assign the weights as $w(p_i^k \to p_i^\ell) = w(C_\ell)$ and $w(p_i^k \leftarrow p_i^\ell) = w(C_k)$. (See also Figure 4.5)

Intuitively, we can think of this assignment of weights as follows. We only pay the cost of a color when entering its vertex – all consecutive usage is free.

Min-Color Separator as Shortest Path on H

From Lemma 32, we know that every color separator S has a separating cycle γ in G^* that is well-behaved. Recall that a well-behaved separating cycle γ comes with a mapping $\mathcal{M}$ such such that all occurrences of a color in the color-cycle $\mathcal{M}(\gamma)$ are consecutive. Let $\mathcal{C}^*$ be the set of all colors in $\mathcal{M}(\gamma)$. Suppose we define weight of γ as $w(\gamma) = \sum_{C_j \in \mathcal{C}^*} w(C_j)$. Since $\mathcal{C}^* \subseteq S$, we have $w(\gamma) \leq w(S)$. Therefore, it suffices to compute a minimum weight well-behaved separating cycle γ in G^*.

Since γ is a cycle in the dual graph G^*, we can assume that one of the vertices say $v_i^* \in \gamma$ is given to us. That is we want to find the shortest well-behaved separating cycle in G^* that passes through v_i^*. Indeed, we can repeat for each vertex of G^* and return the minimum, which will be the shortest well-behaved separating cycle.

We modify H by adding two special vertices v_{src} and v_{dst}. We add the edges $(v_{src} \to a_i^j)$ and assign its weight to be $w(C_j)$. Here a_i^j is the set of L_a vertices corresponding to v_i^* and $C_j \in \chi(v_i^*)$. Similarly, we connect all L_b vertices corresponding to v_i^* as $(b_i^j \to v_{dst})$ and assign a weight of zero to these edges.

Lemma 39 *Any path π from v_{src} to v_{dst} in H with weight $w(\pi)$ corresponds to a separating cycle γ of weight $w(\gamma) = w(\pi)$ passing through v_i^*.*

Proof: Given a path π, we build a separating cycle γ as follows. Every clique edge $(p_i^k \to p_i^\ell)$ corresponds to switching from color C_k to C_ℓ at vertex v_i^*. Every intra-layer free edge such as $(a_x^j \to a_y^j)$ corresponds to moving from vertex v_x^* to v_y^* using color C_j. Every inter-layer free edge such as $(a_x^j \to b_y^j)$ corresponds to moving from vertex v_x^* to v_y^* using color C_j but also crossing π_{st} once. Since v_{src} and v_{dst} are both connected to copies of v_i^* but in different layers, we obtain a cycle γ in the dual graph that crosses π_{st} an odd number of times. This holds because, observe that every time the path π takes an inter-layer edge, it crosses π_{st} in G^*. Since v_{src} and v_{dst} are in different layers, the path π

must take an odd number of inter-layer edges and therefore crosses π_{st} an odd number of times. It is easy to verify that $w(\gamma) = w(\pi)$.

Given a separating cycle γ, we can first obtain a well-behaved mapping $\mathcal{M}$ using Lemma 32. Now we build a path π in H as follows. If $e^* = (v_x^*, v_y^*)$ be an edge in γ such that mapping $\mathcal{M}(v_x^*) = \mathcal{M}(v_y^*) = C_j$ and e^* does not cross π_{st}, we add intra-layer free edges such as $a_x^j \to a_y^j$ to our path. If e^* does crosses π_{st}, we add inter-layer free edges such as $a_x^j \to b_y^j$ to our path. The more interesting case is when $\mathcal{M}(v_x^*) = C_k$ and $\mathcal{M}(v_y^*) = C_\ell$. In this case, we know from Lemma 30 that $C_\ell \in \chi(v_x)$ and we can add the clique edge $a_x^k \to a_x^\ell$ followed by the free edge $a_x^\ell \to a_y^\ell$. It is again easy to verify that the path π in H we obtained has weight $w(\pi) = w(\gamma)$. ■

Using Lemma 39, we can now simply compute a shortest path from v_{src} and v_{dst} for each v_i^* and finally return the separating cycle corresponding to the choice of v_i^* that minimizes the shortest v_{src} and v_{dst} path. We conclude with the following theorem.

Theorem 14 *The minimum color separator on color-connected planar graphs can be computed in polynomial time.*

Chapter 5

Shortest Paths with Removable Obstacles

In this chapter, we turn our attention to the problem of computing shortest path from a given source to destination in presence of 'pairwise disjoint' removable obstacles in the plane. In other words, given a set of disjoint polygonal obstacles in the plane and an integer parameter k, which k obstacles should we remove to obtain the shortest obstacle-free path between two points s and t? Equivalently, what is the shortest path that is allowed to violate (pass through) up to k obstacles?

We call a path violating at most k obstacles a *k-path*, generalizing a traditional obstacle-free path, which is a 0-path. We assume a polygonal environment P containing h disjoint convex obstacles in the plane, with a total of n vertices, all lying inside a rectangle R (the outer boundary). The complement of the obstacles within R is called *free space*. Given a fixed source point s in free space, we want to compute shortest k-paths, for $k \leq h$, to all other points of free space. The description of these shortest paths can be compactly encoded as a finite partition of the plane, called the *shortest k-path map*. We use the notation $\pi_k(t)$ to denote the shortest k-path from s to t, with the fixed source s being

implicit, and denote the length of this path by $d_k(t)$.

In this chapter, we investigate structural and computational aspects of shortest k-paths. The problem differs from the 0-path problem in nontrivial ways even in the plane. In particular, two shortest 0-paths originating at a common source cannot intersect, by the triangle inequality, and this non-crossing property of 0-paths is an essential ingredient for computing them in optimal time [30]. In contrast, two shortest k-paths can cross each other, for any $k > 0$. Our approach to solving the k-path problem is to compute a *shortest k-path map* SPM_k, which is a partition of the plane into equivalence classes of cells (regions), where all destination points inside a cell have the same combinatorial structure of shortest k-paths to s. Once the map is known, the shortest k-path to any destination can be computed by performing a point location query on the map [31, 32].

This *obstacle-removing* shortest path generalizes the classical obstacle-*avoiding* shortest path problem, by giving the planner an option of essentially "tunneling" through obstacles. Besides an interesting problem in its own right, it is also a natural formulation of tradeoffs in some motion planning settings. For instance, it might be beneficial to remove a few critical blockages in a workspace to significantly shorten an often traveled path, just as an urban commuter may strategically pay money to use certain toll roads or bridges to avoid traffic obstacles. In general, our model with removable obstacles is useful for applications where one can adapt the environment to enable better paths such as urban planning or robot motion planning in a warehouse setting.

Results and Chapter Organization

We show that SPM_k has $O(kn)$ regions and $O(kn)$ edges and that this bound is tight. We present an $O(k^2 n \log n)$ time and $O(kn \log n)$ space algorithm for computing SPM_k using the continuous Dijkstra framework, which constructs each SPM_j for $0 \leq j \leq k$

sequentially. The running time of the algorithm is optimal for $k = O(1)$.

The rest of this chapter is organized as follows. In Section 5.1, we discuss some properties of k-paths that are central to the algorithms presented later in the chapter. In Section 5.2, we discuss some relevant background for shortest path maps and prove a tight bound on the size of map defined by shortest k-paths. Finally in Section 5.3, we give an algorithm to compute the map.

5.1 Properties of k-paths

Given a point p in free space, a shortest *k-path* $\pi_k(p)$ connects s to p, crosses the interiors of at most k obstacles, and has minimum length among all such paths. On occasion, we also need to reason about paths crossing *exactly* k obstacles, and we refer to such a path as an $(= k)$-path. We begin with the easy observation that the problem can be solved in polynomial (quadratic) time, using a Dijkstra-like search on a "visibility graph."

Theorem 15 *Given a polygonal domain P with h convex obstacles and n vertices, a source point s and a destination t, we can compute a shortest k-path from s to t in worst-case time $O((kn + h^2)\log n + kh^2)$.*

Proof: By the triangle inequality, each edge of the shortest path $\pi_k(t)$ is either an edge of an obstacle polygon or is a tangent between two obstacles, where we include tangents from s and t. (Each pair of convex obstacles has four tangents.) Let V_1 be the set of obstacle vertices (including s and t) and E_1 the set of all polygon edges and the tangents. Each edge of E_1 is assigned a weight equal to its Euclidean length, and has a label equal to the number of obstacles it crosses. We can compute the set E_1, along with the labels, in time $O(n + h^2 \log n)$ [33]. We now construct a graph $G = (V, E)$,

with $O(kn)$ vertices and $O(n + kh^2)$ edges, as follows. For every $v \in V_1$, we create $k + 1$ copies $v_0, v_1, \ldots, v_k$, corresponding to the number of obstacles crossed on the path to v. For every edge $(u, v) \in E_1$ that passes through $j \leq k$ obstacles, we add the edges $(u_0, v_j), (u_1, v_{j+1}), \ldots, (u_{k-j}, v_k)$. We create two new vertices s and t and connect them to their respective copies in G. That is, s connects to $s_0, s_1, \ldots, s_k$ and t connects to $t_0, t_1, \ldots, t_k$ with zero weight and zero crossing edges. The shortest path from s to t in this graph is the shortest k-path, and the claimed bound follows. ■

The visibility graph-based approach is inherently quadratic in the worst case, because the number of obstacles can be $h = \Omega(n)$. It also is limited to computing the shortest k-path to only one point (or a fixed set of points) at a time, although it can be extended to support queries in $O(h(k + \log n))$ time apiece after quadratic preprocessing.

The main result of this chapter is an algorithm to compute shortest k-paths from s to all points of free space in *subquadratic* time $O(k^2 n \log n)$. We do this by computing a *shortest k-path map* of free space; we also prove a tight bound of $\Theta(kn)$ on the combinatorial complexity of SPM_k. Note that the length of a shortest k-path to a point is unique, although some points (along bisectors forming the boundaries of regions in the shortest path map) can be reached by multiple shortest k-paths. For simplicity, however, we assume that the obstacles are in general position, so that the shortest k-path to each obstacle vertex is unique. (Otherwise, if a vertex is reached from s by multiple shortest k-paths, we pick one of them arbitrarily.)

We begin by highlighting a conceptual difficulty with shortest k-paths. The shortest paths to two different destinations can cross each other, which poses an inherent difficulty for the continuous Dijkstra framework of geometric shortest paths [30], since that method depends on the fact that two Euclidean shortest paths from a common source cannot intersect.

Lemma 40 *There exist obstacle configurations such that for two destinations t_1, t_2 in free space, the shortest k-paths $\pi_k(t_1)$ and $\pi_k(t_2)$ cross each other, for $k > 0$.*

Proof: The construction, shown in Figure 5.1, has two identical obstacle bundles A and B placed parallel to the y-axis. Each bundle contains four vertical strips with perforations (single-point openings that split the original strip into disjoint sub-strips). The horizontal spacing between the strips in a bundle is infinitesimal, but for clarity the strips are shown separated in the figure. The points s and t both lie on the x-axis at distance 1 to the left and right of bundles A and B, respectively. We show that there are two shortest 1-paths from s to t, which cross each other, as shown in the figure. We then conclude that by perturbing t up and down slightly we obtain two destination points t_1 and t_2 with their shortest 1-paths crossing, as claimed.

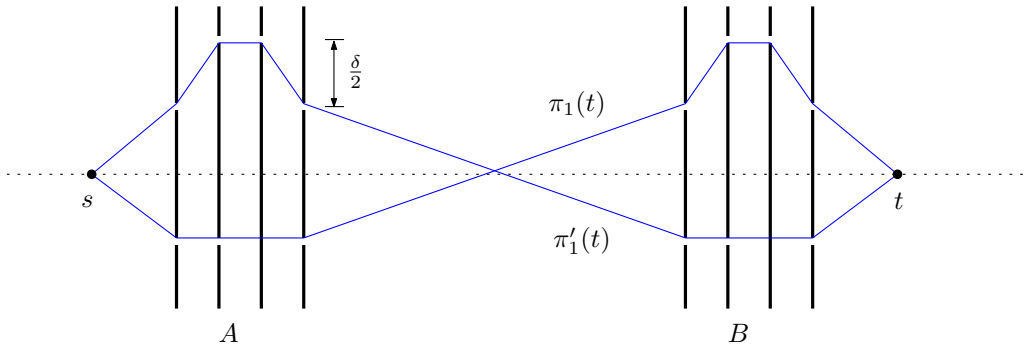


Figure 5.1: Two intersecting 1-paths.

Within each bundle, the openings form an *upper* and a *lower* group. In the upper group, strips 2 and 3 have an opening at $y = (1 + \delta/2)$, and strips 1 and 4 have openings at $y = 1$. In the lower group, all except strip 3 have an opening at $y = -1$. If the distance between the bundles is D, then a shortest 0-path has length $2\sqrt{2} + D + 2\delta$, and a shortest 2-path has length $2\sqrt{2} + D$. A path with exactly one crossing in an upper group has length at least $2\sqrt{2} + D + 3\delta/2$, and a shortest path with one crossing in a lower group has length $2\sqrt{2} + \sqrt{D^2 + 4} + \delta < 2\sqrt{2} + D + 2/D + \delta$. By choosing $D = 10$, say, and

$\delta = 4/D$, we can force a shortest 1-path to go through exactly one group of each type. This gives two intersecting shortest k-paths, $\pi_1(t)$ and $\pi_1'(t)$. Now, let t_1 (resp. t_2) be a destination point obtained by shifting t vertically up (resp. vertically down) infinitesimally. Then it is easy to see that the shortest 1-paths $\pi_1(t_1)$ and $\pi_1(t_2)$ cross each other. ■

Fortunately, as we show in this section, shortest k-paths can always be decomposed into appropriate non-crossing subpaths to which the continuous Dijkstra method can be applied, working on multiple copies of free space connected using the metaphor of a *k-level garage*. Toward that goal, we establish a series of lemmas.

Lemma 41 *A shortest path with exactly k crossings can be decomposed into a shortest path with exactly $(k-1)$ crossings, a straight line segment inside an obstacle, and a shortest path with zero crossings.*

Proof: Let $\pi = (v_1, v_2, \ldots, v_m)$ be an $(= k)$-path from v_1 to v_m. Going backward from v_m along π, let v_i be the first vertex such that the segment $\overline{v_{i-1}v_i}$ intersects one or more obstacles. Let H be the obstacle that is closest to v_i along the segment $\overline{v_{i-1}v_i}$. By the convexity of H, the segment $\overline{v_{i-1}v_i}$ intersects H at two points, which we call p and q, and the segment $\overline{pq}$ lies entirely within H. By subpath optimality, the path from v_1 to p is a shortest path with exactly $k-1$ crossings; by construction, the segment $\overline{pq}$ lies inside the obstacle; and the subpath from q to v_m crosses no obstacles. ■

Observe that for any shortest k-path π, the subpath between any two consecutive vertices v_{i-1} and v_i of π is the straight line segment $\overline{v_{i-1}v_i}$. Since the part of π that lies inside an obstacle H must be coincident with one such segment, we have the following.

Corollary 16 *In a shortest k-path, the path segments preceding and following any obstacle crossing are collinear with the path segment inside the obstacle.*

Lemma 41 allows us to break any $\pi_k(t)$ into a $(k-1)$-path $\pi_{k-1}(p)$, a subpath line segment $\overline{pq}$, and an obstacle-free subpath between q and t. We label the last two subpaths

with the number of obstacles crossed by the prefix of the path, and call these labels the *prefix counts*. In particular, the prefix count for the subpath $\overline{pq}$ is $k-1$, and the prefix count for the subpath from q to t is k. By a recursive application of Lemma 41, we can decompose $\pi_k(t)$ into $2k+1$ disjoint subpaths whose labels are in non-decreasing order.

The key consequence of this decomposition is the following lemma, which says that subpaths with the same prefix count cannot cross. The example in Figure 5.1 is consistent with the lemma, because the intersecting edges of the two crossing shortest k-paths have different prefix counts.

Lemma 42 *Let $\pi_k(t)$ and $\pi'_k(t')$ be two subpaths whose prefix counts are the same. Then $\pi_k(t)$ and $\pi'_k(t')$ do not cross each other.*

Proof: The proof follows from a simple application of the triangle inequality: if two subpaths with the same prefix count intersect, then we can reconnect the prefix of each path to the suffix of the other, and possibly perform a local shortcut, either shortening at least one path or leaving them the same length but without a crossing. Since the intersecting subpaths are either both inside some obstacle or in free space, avoiding the intersection does not increase the number of obstacle crossings for either path. ■

The next two lemmas establish properties of shortest k-paths that will be useful later.

Definition 13 *A point p is k-*visible *from the source s if the segment $\overline{sp}$ passes through at most k obstacles. A k-*visibility edge *is a shortest k-path with exactly one edge.*

Lemma 43 *If p is not $(k-1)$-visible from s, then the path $\pi_k(p)$ must be an $(=k)$-path.*

Proof: By contradiction. Suppose $\pi_k(p)$ passes through fewer than k obstacles. Since p is not $(k-1)$-visible from s, $\pi_k(p)$ must have at least one bend. The path can then be shortened by going through the obstacle causing this bend, thereby increasing

the number of crossings by 1. The resulting path is shorter than $\pi_k(p)$ and has at most k crossings, contradicting the optimality of $\pi_k(p)$. ■

Let $d_k(p)$ be the length of a shortest k-path to a point p. Clearly, a path that crosses j obstacles and contains at least two segments can be made even shorter if it is allowed to pass through more obstacles. Thus, it follows that for any point p that is not $(k-1)$-visible from s, we must have $d_j(p) > d_{j+1}(p)$, for $j < k$.

Lemma 44 *For any point p that is not $(k-1)$-visible from s, the lengths of the shortest j-paths form a decreasing sequence:*

$$d_0(p) > d_1(p) > \ldots > d_i(p) > \ldots > d_k(p)$$

5.2 Shortest Path Map SPM_k: Properties and Bounds

Having established the basic properties of shortest k-paths, we now begin our discussion of the shortest k-path map SPM_k.

Definition 14 *Given a shortest k-path $\pi_k(p)$, we define the* k-predecessor *of p to be the vertex of P (including s) that is adjacent to p in $\pi_k(p)$. The partition of free space into connected regions with the same k-predecessor is called the* shortest k-path map*, and denoted SPM_k. The subset of SPM_k for which the shortest path $\pi_k(p)$ to every point p has exactly k crossings is called the shortest $(= k)$-path map and denoted by $SPM_{=k}$. See Figure 5.2 for an example.*

Unlike SPM_0, in which the predecessor of a region is always inside or on the boundary of the region, the predecessor of a region in SPM_k may lie outside the region. Moreover, multiple regions in SPM_k may have the same predecessor. (See Figure 5.2.) Thus, we need to maintain additional information with polygon vertices to disambiguate the predecessor

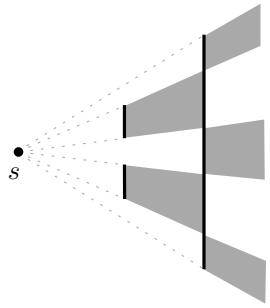


Figure 5.2: The shaded region denotes the cells of SPM_1 for which the 1-predecessor is $(s, 0)$. Note that unlike SPM_0, there are multiple cells with the same predecessor.

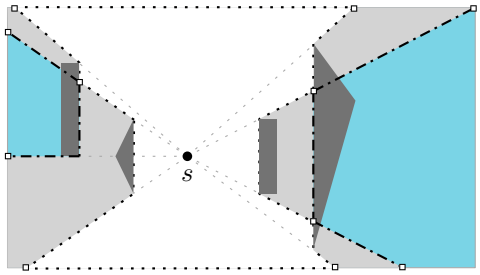


Figure 5.3: The boundary ∂V_1 of the region V_1 is dash-dotted, and it encloses the boundary ∂V_0, which is shown with dotted segments. The region V_0 is shown in white, $V_1 \setminus V_0$ is shown shaded gray. The blue region denotes $V_2 \setminus V_1$.

relation. In particular, let v be the *k-predecessor* of p, namely, the vertex adjacent to p in $\pi_k(p)$. Suppose the line segment $\overline{vp}$ crosses $(k - i)$ obstacles, for some $0 \leq i \leq k$. Then the length $d_k(p)$ of $\pi_k(p)$ is the sum of the length of the i-path to v and the length of segment $\overline{vp}$. We need to maintain the values $d_i(v)$ for all obstacle vertices v and all integers $i = 0, 1, \ldots, k$. In other words,

> For a point p in $SPM_{=k}$, we identify the k-predecessor of p by the pair (v, i), where v is a vertex of P and $i \in \{0, 1, \ldots, k\}$, such that $d_k(p) = d_i(v) + |\overline{vp}|$ and the segment $\overline{vp}$ crosses $(k - i)$ obstacles.

Thus, the total number of k-predecessors is $O(kn)$. However, this alone does not bound the number of regions in $SPM_{=k}$ because multiple regions can have the same k-predecessor and the same crossing sequence. Toward our goal of bounding the combinatorial complexity of the map, let us begin with the notion of k-visibility.

We define V_k to be the region consisting of k-visible points, which is star-shaped and therefore simply connected (Figure 5.3). Now if $\pi_k(p)$ crosses fewer than k obstacles, then by Lemma 43, p must lie in V_{k-1}. The path $\pi_k(p)$ is a straight line segment and the k-predecessor of p is s. Therefore, we have the following.

Lemma 45 *All points p such that $\pi_k(p)$ has fewer than k crossings lie in V_{k-1}. Outside of V_{k-1}, SPM_k is the same as $SPM_{=k}$, the shortest path map with exactly k crossings.*

This simplifies our discussion and allows us to decompose SPM_k into two distinct regions, V_{k-1} and $SPM_{=k}$. In the following, we study structural properties of these regions and use them to compute upper bounds on their respective sizes. Later, we combine them to compute an upper bound on the size of the map SPM_k.

5.2.1 k-Visibility Region

We first bound the complexity of the boundary of V_k, the region visible from s by a segment crossing at most k obstacles.

Lemma 46 *The number of edges on the boundary ∂V_k is $O(n+h) = O(n)$.*

Proof: Every vertex of ∂V_k is either a vertex of P or a projection of one of the $2h$ tangents from s to an obstacle of P. The edges on the boundary ∂V_k are therefore sub-segments of the tangents or parts of obstacle boundaries. Each projection vertex belongs to a segment of ∂V_k collinear with s, and the endpoint x farther from s is the end of a maximal segment $\overline{sx}$ that crosses exactly k obstacles. Therefore, each of the $2h$ tangents gives rise to at most one segment of ∂V_k and at most two vertices. ■

More interestingly, the bound on the total complexity of these regions is less than the sum of the individual bounds.

Lemma 47 *The total number of edges on all ∂V_i, for $0 \leq i \leq k$, is $O(n+hk)$.*

Proof: Any vertex v of P belongs to ∂V_i for at most one value of i, namely the i (if any) such that $\overline{sv}$ intersects exactly i obstacles. For $j < i$, v is outside ∂V_j, and for $j > i$, v is in the interior of ∂V_j. There are $O(h)$ edges of ∂V_i (for any i) not incident to a vertex of P. Summing over all $i \leq k$ completes the proof. ■

By connecting s to all vertices on boundary ∂V_{k-1}, we can easily decompose V_{k-1} into constant complexity regions in SPM_k.

5.2.2 The k-Level Garage and the Structure of $SPM_{=k}$

We now introduce our main idea for computing the shortest k-path map. By Lemma 41, an $(= k)$-path from s to a point p is the concatenation of a $(k-1)$-path to the boundary of some obstacle H, a shortest path inside H, and a shortest path in free space from the other side of H to p. This suggests an incremental construction of $SPM_{=k}$ from $SPM_{=(k-1)}$. We describe this construction using the metaphor of a k-level *parking garage* with elevators.[1] The idea is to create multiple copies of the input polygonal domain and stack them in levels such that the *shortest paths at each level have the same prefix count and therefore do not intersect.* The planar subdivision of free space at the top level is $SPM_{=k}$.

Definition 15 (k-garage) *We construct the* k-garage structure *by stacking* k *copies (or floors) of the input polygonal domain* P *on top of one another, with special connections at the obstacle boundaries. We connect the obstacle* H *on floor* i *to its counterpart on floor* $i+1$ *such that any path that enters* H *on floor* i *can exit only on the next higher floor—in a sense, obstacles act as elevators.*

Our algorithm to construct $SPM_{=k}$ makes use of the *continuous Dijkstra method*, which simulates the expansion of a unit speed wavefront from the source s in free space. The wavefront at time T contains all points p whose shortest path distance from s is T. The boundary of the wavefront is a set of circular arcs called *wavelets*, each generated by an obstacle vertex (including s) already covered by the wavefront. The generating vertex

[1]The garage metaphor is also used in the context of finding homotopically different paths in [34], but the properties and technical details of our k-garage are quite different.

v is called the *generator* of the wavelet and is identified by the pair (v, w), where w is the time at which v was reached by the wavefront. Since the wavefront moves at unit speed, w is precisely the length of the shortest path from s to v. The generators can be thought of as sources *additively weighted* with delays, since they start emitting wavelets at time w after the start of the simulation. The locus of the meeting points of two adjacent wavelets is a *bisector* curve. Taken together with the obstacle boundaries, bisector curves partition free space into regions of the shortest path map.

We extend the continuous Dijkstra method to our k-garage structure. Each level of the garage is a plane with polygonal obstacles on which wavefronts propagate as usual, but the wavelets can now move to higher floors by entering the obstacles (elevators). More precisely, when the wavefront hits an obstacle H, it is absorbed by the outer boundary of H and is immediately re-emitted into the interior of H. When that wavefront reaches the inner boundary on the other (previously unreached) side of H, it is absorbed and immediately re-emitted on the next higher floor of the garage. This vertical movement therefore adds no delay. In this modified setting, the wavefront at time T contains points on all floors that are at distance T from the source.

The region V_{k-1} is removed from the polygonal domain on floor k of the k-garage because the shortest k-path is known for every point p in V_{k-1}—it is simply the line segment $\overline{sp}$—and leaving these points in the polygonal domain on floor k would create redundant copies of this path. We defer the exact details of our algorithm to Section 5.3. In the following, we note some properties of the k-garage structure useful to our algorithm.

1. If π is a shortest s–t path from s on floor 0 to t on floor k, then the downward projection $\pi^{\downarrow}$ of π, obtained by projecting π into the planar domain P, is a shortest k-path to t. (To see this, suppose for contradiction we have another k-path π_c from s to t that is shorter. Then by applying Lemma 41 recursively, we can break π_c

into $2k + 1$ disjoint subpaths ordered by their prefix counts. We now lift the paths into the levels of the garage and concatenate them in order: if the prefix counts of the current and the next subpath are the same, join their common endpoint at the same level as the prefix count; otherwise join their common endpoint at the next level. This transforms the path π_c into a shortest path $\pi_c^{\uparrow}$ from s on floor 0 to t on floor k. Since the vertical movement between the garage floors incurs no delay, the lifted path $\pi_c^{\uparrow}$ is shorter than π, which is a contradiction.)

2. Since wavefront propagation on floor i is affected only by wavelets coming from floors below it, we can think of wavefront propagation on floor i as occurring in a polygonal domain with *multiple* sources. On floor $i > 0$, all sources correspond to generators of wavelets coming from lower floors.

3. To compute the sources at floor $i > 0$, we need to consider only wavelets coming from floor $i - 1$. This follows from Lemma 44, which implies that even if wavelets were allowed to ascend multiple floors in an elevator, a wavelet from floor $i - 1$ would reach floor i no later than the wavelets from other lower floors.

4. The planar subdivision formed by bisectors of colliding wavelets on floor i is the shortest path map for $(= i)$-paths, $SPM_{=i}$. Note that since the obstacles are convex, a shortest path to a point on floor i cannot cross the same obstacle (on any floor) more than once, or else it can be made even shorter.

This suggests a natural way of computing the shortest path map $SPM_{=k}$. We construct maps $SPM_{=i}$ for $i = 0, 1, \ldots, k$ iteratively. Each iteration $i > 0$ is defined by ordinary shortest path propagation with a set of sources that come from the previous iteration. In the following section we use these observations to compute a bound on the size of the shortest k-path map SPM_k.

5.2.3 Complexity of SPM_k

The shortest k-path map SPM_k on the top floor of the k-garage is precisely $SPM_{=k}$ in the portion of free space that is outside V_{k-1}, as shown in Lemma 43. The boundary of V_{k-1} has linear size, and so we only need to bound the complexity of $SPM_{=k}$. To bound the complexity of $SPM_{=k}$, we consider the embedded planar graph G_k formed by $SPM_{=k}$, V_{k-1}, and the obstacle polygons. We note the following property of planar graphs, which is a direct consequence of Euler's formula.

Lemma 48 *Let f be the number of faces in a planar graph $G = (V, E)$. If all the vertices of G have degree three or more, then the size of G is $O(f)$.*

Proof: Let $d(v)$ be the degree of a vertex v. Since $\sum_{v \in V} d(v) = 2|E|$, and $d(v) \geq 3$, we have $2|E| \geq 3|V|$. Substituting this in Euler's formula $|V| - |E| + f = 2$ gives us $|V| \leq 2f - 4 = O(f)$. Since $|E| = |V| + f - 2$, we conclude that $|V| + |E| = O(f)$. ■

Observe that the "interesting" vertices in G_k are the points where bisectors meet obstacle boundaries or meet each other, and therefore have degree at least three. If f is the number of faces, then by Lemma 48 the complexity of the map due to these vertices is $O(f)$. In addition to this, G_k can also have $O(n)$ vertices of degree two corresponding to the vertices of obstacle polygons, giving a total complexity bound of $O(f + n)$.

Therefore, in order to compute a bound on the complexity of $SPM_{=k}$, it suffices to bound the number of faces f in the graph G_k. We begin with the following well-known result [30].

Lemma 49 *The shortest path map of m sources weighted by their delays in a polygonal domain with n vertices and h holes has $f \leq m + n + h \leq m + 2n$ faces. By planarity, the total complexity of the map is $O(f + n)$.*

The key to the proof of the preceding lemma is that each shortest path map region is star-shaped and connected to the predecessor of all points in the region. Since the total

number of predecessors is at most $(m+n)$, the number of faces due to these regions is also at most $(m+n)$. Crucially, this lemma does not immediately apply to $SPM_{=k}$, because some predecessors of regions on the k^{th} floor belong to regions *below the* k^{th} *floor*. That is, some of the m sources are not in the polygonal domain, so the argument that each region is connected to its predecessor does not hold. Fortunately, the argument of Lemma 49 is a topological one, and we can create a topological domain in which the argument applies.

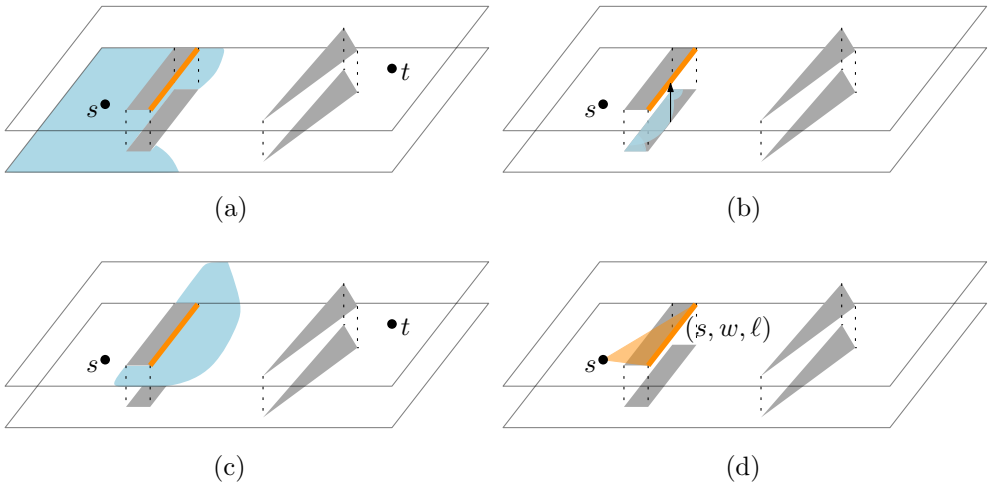


Figure 5.4: (a)–(c) An example illustration of wavefront propagation across garage floors. The wavefront ascends between floors by entering into obstacles (elevators) and creates boundary sources at the next level. We continue wavefront propagation at the next level using these boundary sources. (d) Creating a pseudo-polygonal domain by connecting a source on a higher level to its predecessor on an earlier level by a triangular "flap".

Every point $p \in \partial P$ outside of V_{k-1} is labeled by a $(k-1)$-crossing distance $d_{k-1}(p)$. If p belongs to an obstacle H, and there exists some $q \in \partial H$ such that $d_{k-1}(q)+|\overline{qp}| < d_{k-1}(p)$, then $\pi_k(p)$ may reach p by passing through H. The wavefront that determines $SPM_{=k}$ will be initialized with a weighted source that reaches p by "elevator" passing through H. If $q \in \partial H$ minimizes $d_{k-1}(q) + |\overline{qp}|$, then the predecessor of q on $\pi_{k-1}(q)$ is the generator of the wavelet that first reaches p in the wavefront. We partition each edge of ∂H into maximal sub-edges with the same predecessor. For each sub-edge with predecessor v, we construct a triangular "flap" by drawing the segments from the sub-edge endpoints to

v. Shortest paths propagate from v toward the k^{th} garage floor inside the flap, and in the pseudo-polygonal domain obtained by gluing all the flaps onto the boundary of free space, each shortest path map region is connected to its predecessor. If these flaps were projected into the plane, they would likely overlap, but topologically they do not alter the structure of the domain, and they add only two edges per flap.

Lemma 50 *Let P be a polygonal domain with n vertices and h holes. If P is extended by gluing at most m triangular flaps to its boundary, then the shortest path map of m sources weighted by their delays in this extended polygonal domain has $f \leq m+n+h \leq m+2n$ faces and total complexity $O(m+n)$.*

The preceding lemma applies to the propagation of shortest paths on each floor of the k-garage and also to propagation inside the obstacles (elevators). In both cases the key to bounding the complexity of an iterated construction is bounding the number of sources that propagate into the next level, whether elevator or garage floor. In each elevator and on each garage level $i > 0$, the sources are located on the domain boundary. For simplicity we partition the sources at obstacle vertices, so each source is a maximal (sub-)edge ℓ on some obstacle boundary ∂H, with an associated generator (v, w). We refer to such a source as a *boundary source* and represent it by the triple (v, w, ℓ). Shortest paths from a source (v, w, ℓ) enter the domain through edge ℓ, and their predecessor is vertex v with weight (delay) w. As noted above, each boundary source defines a triangular flap glued onto the boundary of the propagation domain; the flap is the convex hull of ℓ and v.

When boundary sources propagate into some domain (either P or the interior of an obstacle), they define a shortest path map S in the domain. We say that if the region of S corresponding to a source $s = (v, w, \ell)$ intersects a domain edge, then s *claims* the intersection interval on that edge. An *entry claim* of a source (v, w, ℓ) is a claim on edge ℓ itself; entry claims can be ignored for further propagation, since a path that

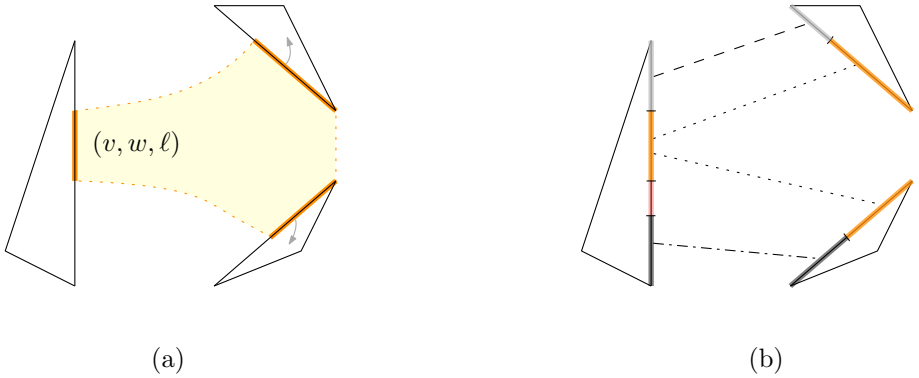


Figure 5.5: (a) Exit claims for the boundary source (v, w, ℓ) need to be propagated to next level. (b) Connecting the sources with their exit claims gives a bipartite planar graph.

enters the domain through ℓ and exits through the same edge can be shortened. *Exit claims* (ones on edges other than ℓ) define the sources for the next level of shortest path propagation. (See Figure 5.5.) Within any edge, a maximal sequence of exit claims with the same source is called an *exit claim cluster*. In other words, exit claims of a source (v, w, ℓ) on an edge e may be disconnected and each connected sequence is precisely an exit claim cluster. Note that these exit claim clusters give rise to the boundary sources for subsequent wavefront propagation. That is, for an exit claim cluster on edge e with source (v, w, ℓ), the corresponding boundary source at the next level is (v, w, ℓ'), where ℓ' is the minimal subsegment of e containing the cluster. As noted, entry claims inside ℓ' do not affect shortest path propagation at the next level.

Lemma 51 *Let S be the shortest path map obtained by propagating m boundary sources into a polygonal domain with n vertices. Then the number of exit claim clusters of S is at most $m + O(n)$.*

Proof: Since S is a partition of the domain, the domain boundary is completely covered by claims, which may be either entry claims or exit claims.

We construct an embedded bipartite planar graph whose nodes are claims on the

domain boundary. Every source (v, w, ℓ) that claims some portion of the domain boundary must have an entry claim on ℓ; otherwise the shortest path propagation from (v, w, ℓ) would not enter the domain. For every exit claim τ claimed by source (v, w, ℓ), we draw an *arc* from segment ℓ to τ, following a shortest path segment across the domain interior. We want to bound the total number of these arcs. Since the shortest paths that define S do not cross, these arcs are non-crossing. (See also Figure 5.5.)

We group arcs into *bundles* whose sources and targets lie on the same pair of domain edges. If we pick one arc from each bundle and regard each domain edge as a node in a planar graph, planarity gives a bound of $O(n)$ on the total number of bundles. This bound on the number of bundles is the first step in bounding the number of arcs.

If a bundle joining edges e and e' has $j > 1$ arcs, we draw $j - 1$ cycles, each one defined by two adjacent arcs and the subsegments of e and e' between their endpoints. The cycles for a single bundle are interior-disjoint, but cycles from different bundles may be nested, one containing the other. Note that cycle boundaries cannot cross—they are composed of obstacle boundaries and noncrossing arcs—so nesting is the only possible relation between cycles that are not interior-disjoint.

If a cycle C contains any obstacle, we split the bundle B containing C between the arcs of C, so neither of the resulting two bundles contains C. We charge the splitting of B to one of the obstacles inside C. We choose which obstacle to charge so as to guarantee that each obstacle is charged at most once. If C contains no cycle nested inside it, we charge an arbitrary obstacle inside C. If C contains other cycles, let C' be one at the outermost level of nesting within C. Cycle C' must have at least one of its bounding edges on an obstacle H contained in the interior of C, because otherwise C' would share both obstacle edges with C, which is impossible by construction. Obstacle H is not contained in any cycle C'' nested inside C, because C'' would necessarily contain C', but C' was chosen outermost. We charge the splitting of B at C to H. Note that H cannot

be charged by any cycle inside C (because it is outside all such cycles) or containing C (because it is inside C and hence shielded from such cycles).

Because there are at most $O(n)$ obstacles, each charged for at most one split, the number of bundles after splitting is still $O(n)$. None of the bundles that remain after splitting contains any obstacle inside the quadrilateral it bounds.

Given a bundle incident to edges e and e', we divide it into two sub-bundles, one consisting of arcs directed from e to e' and one consisting of the oppositely directed arcs. Within each sub-bundle, we identify contiguous runs of arcs with the same source. Each maximal run corresponds to an exit claim cluster, so we will call these runs *arc clusters*. We charge the first and last arc cluster in each sub-bundle to the bundle itself. (There are $O(n)$ such charges.) Crucially, every other arc cluster corresponds to a source that appears only in this bundle, because the arcs before and after it in the sub-bundle confine it and prevent it from claiming edges anywhere else. Hence we can charge each such cluster to the source itself; the source is charged only once.

To recap, we bound the number of exit claim clusters by the number of arc clusters. Arcs belong to bundles, and there are $O(n)$ bundles by planarity. To remove obstacles inside bundles, we split bundles at most $O(n)$ times. We break each bundle into two sub-bundles, and pay explicitly for the first and last arc cluster in each, for a total of $O(n)$. We charge each remaining arc cluster to one of the m sources, charging each source at most once, giving a total bound on the number of arc clusters of $m + O(n)$. ■

We are now ready to bound the complexity of $SPM_{=k}$.

Lemma 52 *The number of faces f_k in $SPM_{=k}$ is $O(n(k+1))$. The complexity of $SPM_{=k}$ has the same asymptotic bound.*

Proof: The proof is by induction. Our goal is to show that there exists a constant C such that the number of faces f_k in $SPM_{=k}$ is at most $Cn(k+1)$ for all $k \geq 0$.

We begin with the inductive step. Let m be the number of exit claim clusters in $SPM_{=(k-1)}$. This is the number of boundary sources in "elevator" propagation across the obstacle interiors, going from level $k-1$ to level k. By Lemma 51, the resulting number of exit claim clusters is $m' = m + O(n)$. But m' is the number of boundary sources in the construction of $SPM_{=k}$, and once again by Lemma 51, the resulting number of exit claim clusters is $m'' = m' + O(n) = m + O(n)$, that is, $m'' \leq m + c_1 n$ for some constant c_1.

To establish the base case, recall that a shortest path map with no crossings (SPM_0) has complexity $O(n)$, which implies that the number of exit claims on its boundary is $O(n)$, i.e., at most $c_2 n$ for some constant c_2. Combining the base case and inductive step, we have shown that the number of exit claim clusters on the boundary of $SPM_{=k}$ is at most $c_2 n + k \cdot c_1 n$. The number of faces of $SPM_{=k}$ is at most equal to the number of boundary sources, which is at most $Cn(k+1)$, for $C = \max(c_1, c_2)$. Lemma 48 establishes the total complexity bound. ■

5.2.4 A Matching Lower Bound

We will now bound the size of SPM_k from below by constructing a map with $\Omega(nk)$ regions. We construct an arrangement of obstacles as shown in Figure 5.6. We start with two obstacle bundles A and B placed parallel to the y-axis. Within each bundle, the horizontal spaces between strips are infinitesimal, but they are shown enlarged for clarity. The source s lies on the x-axis with bundle A placed right next to it. Bundle A consists of $3k$ perforated strips. In the first $2k$ strips, the odd numbered ones have openings at $y = 0$ and the even numbered ones have openings at $y = -0.5$. The next k strips have an opening at $y = 0$. Bundle B is placed at a distance D to the right of A and consists of k strips with no openings.

The last k strips in bundle A ensure that shortest k-paths starting at s must exit

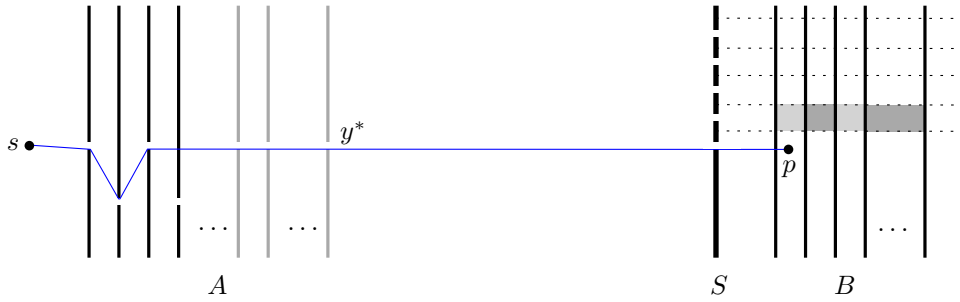


Figure 5.6: A shortest k-path map with complexity $\Omega(nk)$. Bundle A has $2k$ black strips and k gray strips; bundle B has k strips. The thick strip S has $\Omega(n)$ openings. Each opening of S defines k cells in SPM_k, shown shaded (one to the right of each of the k strips in bundle B). A shortest k-path $\pi(p)$ from s is also shown. Observe that since $\pi(p)$ crosses $(k-1)$ strips in bundle A, it can only cross the first strip in bundle B.

from the opening of the last strip in A (denoted by y^*); a path that crosses the last strip in A at some point other than y^* can be shortened while preserving the same number of crossings. Observe that a shortest path starting at s can reach y^* with i crossings, where $0 \leq i \leq k$. However, each crossing avoided results in an additional length of 1 unit. Therefore a shortest path with i crossings at y^* has an additional length of $(k-i)$ units. Also note that a shortest path with i crossings prior to y^* can cross the first $(k-i)$ of the k strips in bundle B, but cannot cross any farther. Therefore, to the right of strip j in bundle B, we get a region with k-predecessor $(y^*, k-j)$ and a total path length (to a point on the x-axis) of $D+j$. This gives us a total of k regions.

We extend this construction to $\Omega(nk)$ regions by adding a vertical strip S, which acts as a *path splitter*. This special strip has a total of m single-point openings at $y = 0, 1, \ldots, m$, denoted by y_i. We place S at an infinitesimal distance to the left of bundle B, creating k new regions for each opening of S. Note that in the range $0 \leq y \leq m$, a path that crosses S other than at one of the perforations y_i can be shortened by detouring through the nearest y_i and inserting one more crossing before y^*. Hence a shortest k-path always passes through one of the y_i. This gives a total of $O(mk)$ regions: the k-predecessor of the region at $y = i$ and to the right of strip j of bundle B will be $(y_i, k-j)$, with a total

path length of $\sqrt{D^2 + i^2} + j$.

The total number of vertices in our construction is $3k \times 4 \ + \ k \times 2 \ + \ (m+1) \times 2 \ = \ 14k + 2m + 2$. By choosing $m = (n - 14k - 2)/2$ and assuming $k < n/28$, we have $m = \Theta(n)$ and the total number of regions in SPM_k is $\Omega(nk)$. This gives us the following lemma.

Lemma 53 *The worst-case complexity of SPM_k is $\Omega(nk)$.*

Combining Lemmas 46, 52, and 53, we get the main result of this section.

Theorem 17 *The shortest k-path map SPM_k has size $\Theta(kn)$.*

5.3 Computing SPM_k

In this section we describe an $O(k^2 n \log n)$ algorithm to construct SPM_k. Recall from our discussion about the k-garage (Definition 15), we can construct $SPM_{=k}$ iteratively, one level at a time. To compute the map at each level, we propagate the sources from the previous level and then perform wavefront propagation at the current level. For this, we use the algorithm for shortest paths in the presence of polygonal obstacles by Hershberger and Suri [30] as a subroutine. Except for a few small modifications required for our setting, most of the algorithm carries over unchanged. In the following, we briefly review the key ideas and discuss the necessary modifications.

The Hershberger–Suri algorithm uses the continuous Dijkstra method, which simulates the propagation of a unit speed wavefront in free space. The wavefront is a collection of circular wavelets. It changes its shape as it propagates and hits obstacles. Each wavelet originates at a *generator*, which may be a point source or an obstacle vertex (an *intermediate source*). A generator for a wavelet γ is identified by the pair (v, w), where v is an input vertex and w is the time at which v starts emitting γ. The Hershberger–Suri

algorithm simulates wavefront propagation over a planar subdivision called the *conforming subdivision* of free space. For each subdivision edge e, and every point $p \in e$, the algorithm identifies the generator whose wavelet first reaches p. Combining these results for all $p \in e$ gives the *wavefront for* e. The key idea of the algorithm is to localize interesting events (such as wavelet collisions) within a constant number of cells in the subdivision. Each free-space edge e of this subdivision is contained in the union of a constant number of cells, called its *well-covering region* $\mathcal{U}(e)$. The wavefront for edge e is computed by combining and propagating the wavefront inside of $\mathcal{U}(e)$. The computed wavefronts are then merged to compute the shortest path map. This is the main result relevant to our algorithm:

Lemma 54 ([30]) *Given a set of polygonal obstacles with n vertices and a set of $O(n)$ sources with delays, one can compute the shortest path map in $O(n \log n)$ time and $O(n \log n)$ space.*

From the discussion preceding Lemma 51, recall that the sources on floor i are identified by triples (v, w, ℓ), where ℓ is a (sub-)edge of some obstacle H, (v, w) is a weighted point source on some floor $j < i$, and the wavelet γ generated by (v, w) enters floor i from the interior of H (an elevator) passing through edge ℓ. Each source (v, w, ℓ) defines a triangular flap glued onto the boundary of free space at ℓ. Conceptually, we think of the wavelet γ from (v, w, ℓ) as propagating in the flap before it enters floor i. Algorithmically, we can ignore the flap and start the propagation in free space at edge ℓ. This calls for a slight modification in the initialization step of the Hershberger–Suri algorithm. In particular, we do the following for each edge e of the conforming subdivision:

1. Find all boundary sources (v, w, ℓ) such that the well-covering region $\mathcal{U}(e)$ contains ℓ.

2. Initialize *covertime*(e), which is the time at which e would be engulfed by the wavefront, minimizing over all boundary sources (v, w, ℓ) with $\ell \in \mathcal{U}(e)$, and for each such source considering paths from v with delay w, constrained to pass through ℓ.

3. For each source (v, w, ℓ) with $\ell \in \mathcal{U}(e)$, propagate its wavelet γ to e inside $\mathcal{U}(e)$.

In the following lemma we show how to compute the boundary sources for each step of wavefront propagation.

Lemma 55 *Given m boundary sources in a polygonal domain with n vertices, we can compute the exit claims of the sources in $O((m+n)\log(m+n))$ time and space.*

Proof: We apply the Hershberger–Suri algorithm, modified for boundary sources as described above. The algorithm computes the shortest path map for the sources inside the polygonal domain in total time and space $O((m+n)\log(m+n))$. The shortest path map partitions the boundary into $O(m+n)$ intervals (claims), each claimed by its own source.

However, some of these intervals may be entry claims, that is, they are claimed by a source that lies on the same segment. Observe that an entry claim interval must have a non-empty intersection with the interval corresponding to its source. We can therefore identify entry claims by overlaying the set of claim intervals with the boundary sources that form another set of m intervals. Overlaying these two sets of intervals takes additional linear time and space. The remainder is the set of all exit claims, that is, those with a claiming source from a different segment. ■

With these primitives in place, we are ready to describe our algorithm. The input is a polygonal domain P with convex obstacles. We will use M to denote the set of boundary sources passed as input to the Hershberger–Suri algorithm. The algorithm computes two

things: the $(k-1)$-visibility region V and the $(=k)$-path map $SPM_{=k}$, which combined together form SPM_k. The length of the shortest path to any point p can then be easily computed by first locating the region containing p in the map SPM_k and then connecting p to the k-predecessor of this region as described in the beginning of Section 5.2.

Algorithm 5 Algorithm to construct SPM_k

1. Set $M = \{s\}$ and call the Hershberger–Suri algorithm to compute SPM_0 for the polygonal domain P. Initialize V to be the empty region $\emptyset$.

2. Repeat for each $i \in 1, 2, \ldots, k$:

   (a) Using Lemma 55, propagate the sources in SPM_{i-1} through the obstacles in P to compute the set of boundary sources M_{new} for $SPM_{=i}$.

   (b) Identify all the regions in $SPM_{=(i-1)}$ for which the predecessor is s. Observe that this is precisely the region $V' = V_{i-1} \setminus V_{i-2}$. Set P to be the new polygonal domain with this region removed.

   (c) If $V = \emptyset$, then set $V = V'$. Otherwise merge V with V' at the common vertices.

   (d) Set $M = M_{new}$ and call the Hershberger–Suri algorithm to compute $SPM_{=i}$ for the polygonal domain P.

3. Merge $SPM_{=k}$ with V at the boundary of regions of $SPM_{=k}$ that have s as predecessor (i.e. $V' = V_k \setminus V_{k-1}$), to obtain SPM_k.

Observe that after Step 2c of iteration i, the region V is equal to V_{i-1}. Because V_{i-1} contains V_{i-2} and because both regions have linear size (by Lemma 46), Step 2c takes linear time. Therefore, the total running time is dominated by k calls to the Hershberger–Suri algorithm with $O(nk)$ sources (Theorem 17). We have the following result.

Theorem 18 *If P is a polygonal domain bounded by convex obstacles with a total of n vertices, the shortest k-path map for P with respect to a source point s can be computed in $O(k^2 n \log n)$ time and $(kn \log n)$ space.*

5.4 Bibliographic Notes

The problem of computing shortest paths in the presence of obstacles has a long history in computational geometry, dating back to the 1970s. The case of polygonal obstacles in the plane, in particular, has been a subject of intense research [35, 36, 37, 38, 39, 33, 40, 41, 42], culminating in an optimal $O(n \log n)$ time algorithm using the continuous Dijkstra framework [30]. Many other variations of the problem, including shortest paths inside a simple polygon [43, 44, 45], among weighted regions [46], and among curved obstacles [47, 48], have also been studied. The general flavor of our problem is related to geometric optimization where a small number of constraints can be violated. This line of work has been pursued in [49, 50, 51, 52], in the context of low-dimensional linear programming, separability with outliers, and geometric optimization. Our problem can also be viewed as a form of network augmentation, where the goal is to add edges to the network to improve connectivity, diameter, or spanning ratio, etc. [53, 54, 55, 56].

The prior work most closely related to our problem is a recent result by Maheshwari et al. [57], which presents an $O(n^3)$ time algorithm for computing the 1-violation path inside a simple polygon: that is, a shortest path inside a simple n-gon where at most one edge of the path lies outside the polygon. We deal with a different notion of path violation: we compute a k-violation path, for any value of k, in a polygonal domain with n vertices and h convex holes, where the violation count is the number of holes intersected by the path.

Recall that the *minimum constraint removal* problem discussed in Chapters 3 and 4 considers the case where given a set of possibly overlapping obstacles in the plane, one would like to compute the minimum number of obstacles that can be removed to create a path in free space from s to t. An important difference is that here we assume the obstacles to be disjoint, so the existence of a free space s–t path is trivial.

Chapter 6

Shortest Paths with Weighted Obstacle Removal

In the previous chapter, we presented an $O(k^2 n \log n)$ algorithm for shortest paths that pass through (or remove) at most k obstacles. There, we assumed that all obstacles are equivalent in terms of their removal costs. In this chapter, we study a more general formulation where one needs to pay a non-negative cost c_i for removing the i-th obstacle. Formally, let $\mathsf{P} = \{P_1, \ldots, P_h\}$ be a set of h pairwise-disjoint polygonal obstacles in $\mathbb{R}^2$ with n vertices, and let $c_i > 0$ be the cost of removing the obstacle P_i for $i = 1, \ldots, h$. For a path π in $\mathbb{R}^2$, we define its cost, denoted by $c(\pi)$, to be the sum of the costs of obstacles intersecting π, and its length, denoted by $\|\pi\|$, to be its Euclidean length. Given two points $s, t \in \mathbb{R}^2$ and a budget $C > 0$, we wish to compute a path from s to t of minimum length whose cost is at most C.

Indeed, the shortest k-path problem studied in the previous chapter is also an obstacle-removing shortest path where each obstacle removal has cost 1 and the cost budget is k. We will call this the *cardinality* version of the obstacle-removal to distinguish it from the cost-based model of obstacle removal studied in this chapter.

A natural application where the cost model comes in handy is to *path planning under uncertainty*. Imagine, for instance, a workspace with n obstacles, the presence of each obstacle is a random event. That is, the presence of the ith obstacle is determined by a Bernoulli trial with (independent) probability β_i. A natural approach to planning a s–t path in such a workspace is to search for a path that is both short and obstacle-free with high probability. Given a desired probability of success β, we can ask: what is the shortest path from s to t that is obstacle free with probability at least β. This problem is easily transformed into our obstacle removal problem where the obstacle probabilities are mapped to obstacle removal cost, and β is mapped to the cost budget C.

Results and Chapter Organization

We first show that the obstacle-removing shortest path problem is NP-hard for polygonal obstacles in the plane, even if obstacles are vertical line segments by reducing the well-known PARTITION problem to it. This is in contrast with the cardinality version of the problem, which can be solved exactly in $O(k^2 n \log n)$ time [58]. This is discussed in Section 6.1.

The main result of this chapter is a fully-polynomial time approximation scheme (FPTAS) when each obstacle is a *convex* polygon. We first define the notion of the *viability* graph G, which is an extension of the well-known visibility graph [59, 60], for geometric paths that can cross obstacles. Using the viability graph, we present a simple algorithm that returns a path with length at most the optimal[1] but cost at most $(1+\epsilon)C$. The approximation algorithm, while simple, has a worst-case time complexity $\Theta(\frac{n^3}{\epsilon}\,\mathrm{polylog}(n))$. This is discussed in Section 6.2. Then in Section 6.3, we develop a framework for a more efficient and practical approximation algorithm, which also results in a number of related results. Specifically, for any constant $\epsilon > 0$, we can compute

[1]The optimal length is always with respect to the budget C.

a $(1+\epsilon)$-approximate shortest path whose total removal cost is at most $(1+\epsilon)C$ in time $O\left(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon}\right)$, where h is the number of obstacles and n is the total number of vertices in the obstacles. The main idea behind the improvement is to construct a *sparse* viability graph, with only $O(\frac{n}{\epsilon}\log n)$ edges. This approximation scheme immediately gives a corresponding result for the uncertain model of obstacles (see Section 6.5 for this).

The approximation scheme, as a byproduct, also solves the exact L_1 norm shortest path problem in the *cardinality* model of obstacle removal: that is, in $O(kn\log^2 n)$ time, we can decide which k obstacles to remove for the shortest s–t path, which is roughly a factor of k faster than the L_2-norm result from the previous chapter. Alternatively, we can also decide which k obstacles to remove so that the shortest s–t path has length at most $(1+\epsilon)$ times optimal in $O(\frac{kn}{\epsilon}\log^2 n)$ time. This is again faster than the result from Chapter 5 for constant ϵ, if $k=\Omega(\log n)$.

Finally in Section 6.4, we also construct query data structures for answering approximate obstacle removal shortest path queries. If the source s is fixed (one point queries), we construct a data structure of size $O(\frac{nh}{\epsilon^2}\log n)$ such that, given a query point t, it returns a s–t path of length $(1+\epsilon)$ times the optimal with cost at most $(1+\epsilon)C$ in time $O(\frac{1}{\epsilon}\log^2 n + k_{st})$, where k_{st} is the number of edges in the path. The data structure size can be improved to $O(\frac{n}{\epsilon^2}\log n\log\frac{h}{\epsilon})$ if we only return the length of the path. If both points s,t are given in the query (two point queries), the data structure has size $O(\frac{n^2h}{\epsilon^3}\log^2 n)$, and the query time is $O(\frac{1}{\epsilon^2}\log^2 n + k_{st})$. The size of the data structure can also be improved to $O(\frac{n^2}{\epsilon^3}\log^2 n\log\frac{h}{\epsilon})$ if we only return the length of the path.

6.1 NP-hardness

Consider the decision version of the obstacle-removing shortest-path problem: Given a set P of pairwise-disjoint obstacles along with the cost of each object being removed,

two points $s, t \in \mathbb{R}^2$, and two parameters $C, L > 0$, is there a path from s to t of length at most L and cost at most C?

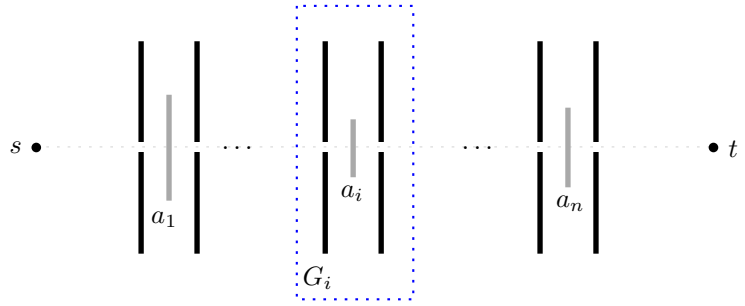


Figure 6.1: Reduction from PARTITION. The gray segment in the obstacle group G_i has length a_i and can be crossed by paying a cost a_i. The tall segments are drawn in black and are placed $\pm\delta$ apart from their corresponding gray segment.

We prove the hardness by a simple reduction from the well-known NP-complete problem PARTITION. An instance of PARTITION is a set of n positive integers $A = \{a_1, a_2, \ldots, a_n\}$, and the problem is to decide whether A can be partitioned into two sets A_1 and A_2 such that $W(A_1) = W(A_2) = \frac{1}{2}W(A)$, where $W(S)$ is the sum of the integers in S. We place the source s at $(0, 0)$ and destination t at $(n+1, 0)$ on the x-axis. We also set $C = \frac{1}{2}W(A)$, $L = \frac{1}{2}W(A) + (n + 1)$ and define a parameter $\delta = \frac{1}{8n}$. For each $i \leq n$ we create a group of obstacles, denoted G_i, which consists of five vertical line segments placed close to each other in the following way. (See also Figure 6.1.)

- The middle segment e_i^m has length a_i, and has its midpoint on the x-axis. The coordinates of its endpoints are $(i, -a_i/2), (i, a_i/2)$. The cost of this obstacle is a_i.
- At x-coordinates $i-\delta$ and $i+\delta$ we place two vertical segments e_i^l and e_i^r symmetrically along the x-axis – each with point-sized holes on the x-axis and length $2(L + 1)$. The point sized holes split the segment e_i^l (resp. e_i^r) into two disjoint *tall* segments e_i^{lu}, e_i^{ld} (resp. e_i^{ru}, e_i^{rd}), of length $(L + 1)$. Each of these segments has cost $(C + 1)$.

Lemma 56 *The set A can be partitioned into two equal-weight subsets if and only if there is a path from s to t of length at most L and cost at most C.*

Proof: The choice of obstacle lengths and costs ensures that any path of length at most L and cost at most C must avoid passing through any of the tall obstacles $e_i^{lu}, e_i^{ld}, e_i^{ru}, e_i^{rd}$ as well as must travel through the point-sized holes between them at all $i = 1, 2, \ldots, n$ groups (otherwise, the path either is longer than L or costs more than C). In each group, however, the path must choose whether to pass through the obstacle e_i^m at cost c_i or travel around it, which increases its length by at least $(a_i - 2\delta)$. Given a valid partition into two subsets A_1 and A_2, finding a path that meets the constraints is easy – just go through short obstacles in groups corresponding to elements in A_1 and go around short obstacles in groups corresponding to elements in A_2. Given a path that meets the constraints, let A_1, A_2 be the elements in A corresponding to short obstacles in groups that the path goes through and goes around respectively. From the cost constraint on this path, it follows that $W(A_1)$ is at most $\frac{1}{2}W(A)$. The length of this path is at least $(n+1) + W(A_2) - 2n\delta = (n+1) + W(A_2) - \frac{1}{4}$. Due to the length constraint, this value can be at most $(n+1) + \frac{1}{2}W(A)$ which gives that $W(A_2)$ is at most $\frac{1}{2}W(A) + \frac{1}{4}$. Note that since the smallest element in A is at least 1, if $W(A_2)$ was more than $W(A)/2$, it must also be more than $\frac{1}{2}W(A) + \frac{1}{2}$. Therefore, we must have $W(A_1), W(A_2)$ to be at most $\frac{1}{2}W(A)$ and since $W(A_1) + W(A_2) = W(A)$, we have a valid partition. ■

We thus obtain the following:

Theorem 19 *Let P be a set of n disjoint polygonal obstacles in a plane, where each obstacle $P_i \in \mathsf{P}$ has an associated removal cost c_i. Given a source and destination pair of points s, t, a removal budget C and a length L, the problem of deciding if there is a s–t path with cost at most C and length at most L is* NP*-hard.*

6.2 A Simple $(1+\epsilon)$-Approximation Algorithm

In this section, we propose a simple polynomial-time approximation scheme for the problem. We begin by noting that an obstacle-removing shortest path only turns at obstacle vertices and crosses the boundary of an obstacle at most twice. While these properties follow easily due to the convexity of P and basic geometry, they are crucial for our algorithms.

The algorithm constructs a *viability graph* $G = (V, E)$, whose nodes are all the obstacle vertices along with s and t. Thus, $|V| = n + 2$. The edges of E correspond to pair of nodes (u, v) for which the line segment uv passes through obstacles of total cost at most C, the cost budget. For each edge $e \in E$, we associate two parameters: cost $c(u, v)$ and length $\|uv\|$, where $c(u, v)$ is the cost of the segment uv. In the worst-case G has $\Theta(n^2)$ edges. It is important to note that the cost of a path π_{st} in a viability graph is defined as the sum of the costs of its edges, whereas the cost of π_{st} in the plane is defined as sum of costs of all obstacles that it goes through. Moreover, the cost of a path in the plane is at most its cost in the viability graph. If the path crosses each obstacle at most once (which is the case for shortest path among convex obstacles), these two costs are the same.

The following algorithm shows how to compute an approximately optimal path in this viability graph. The main idea is that we construct copies of the vertices and the edges of G to convert the multi-objective problem to a single-objective problem.

Let $\kappa = \min\left(\frac{C}{\min_i c_i}, h\right)$. To simplify the approximation error analysis, we first scale all the costs by κ/C, so that the new target cost is κ. We now construct an auxiliary graph $G' = (V', E')$, with $O\left(\left\lceil \frac{2\kappa|V|}{\epsilon} \right\rceil\right)$ nodes and $O\left(\left\lceil \frac{2\kappa|E|}{\epsilon} \right\rceil\right)$ edges, *whose edges only have the length parameter but not the cost parameter*, as follows. We create $\left\lceil \frac{2\kappa}{\epsilon} \right\rceil + 1$ copies $v_0, v_{\frac{\epsilon}{2}}, v_\epsilon, v_{\frac{3}{2}\epsilon} \ldots, v_\kappa$, for each $v \in V$. Then, for each edge $(u, v) \in E$ with cost c and for each $0 \leq i \leq \lceil 2\kappa/\epsilon \rceil$, we add the edge $(u_{i\frac{\epsilon}{2}}, v_{j\frac{\epsilon}{2}})$, where $j \leq \lceil 2\kappa/\epsilon \rceil$ is the maximum

integer with $j\frac{\epsilon}{2} \leq i\frac{\epsilon}{2} + c$. All these edge copies have the same length as edge (u, v)—the cost parameter is now implicitly encoded in the edge copies. Finally we add two new vertices s and t in G' and connect them to all s_i and t_i respectively with zero length edges, for $0 \leq i \leq \lceil 2\kappa/\epsilon \rceil$. We now find the *minimum length* path π from s to t in G', say, using Dijkstra's algorithm, and argue that π is our approximation path.

Theorem 20 *Let* P *be a set of* h *convex obstacles with* n *vertices,* s, t *be two obstacle vertices, and* $C \in \mathbb{R}$ *be a parameter. Let* L^* *also be the length of the shortest* s*–*t *path with cost at most* C*, and let* $G = (V, E)$ *be a viability graph induced by this workspace. If there exists a path* π^* *of length at most* αL^* *with* $\alpha \geq 1$ *and cost at most* C *in the graph* G*, then a* s*–*t *path* π *with length at most* αL^* *and cost at most* $(1+\epsilon)C$ *can be computed in time* $O\left(\frac{\kappa}{\epsilon}(|E| + |V| \log \frac{|V|}{\epsilon})\right)$*, where* $\kappa = \min\left(\frac{C}{\min_i c_i}, h\right)$ *and* $0 < \epsilon < 1$ *is a parameter.*

Proof: First, we construct the auxiliary graph G' as described above. Next, we construct a path π' in G' corresponding to the path π^* in G by mapping edges of π^* to edges in G'. More precisely, let $e = (s, v)$ be the first edge in π^* and let c_e be its cost. Now let $c = 0$ and c' be the value obtained by *rounding down* c_e to the nearest multiple of $\frac{\epsilon}{2}$. We map e to the edge $(s_c, v_{c'})$ in G'. Setting $c = c'$, we repeat the process for all edges in π^*. This gives us the path π' in G' that has the length same as that of π^* (at most αL^*). Clearly, the s–t path π computed using Dijkstra's algorithm on G' must also have length at most αL^*. Moreover, since (scaled) rounded cost of any s–t path in G' is at most κ, the rounded cost of π is also at most κ. Now we only need to bound its original (pre-rounded) cost.

Let C_R be the true (pre-rounded) cost of the path π in the plane and C_A its rounded cost in G'. The approximation error in the cost (due to rounding) is at most $\epsilon/2$ for each obstacle that π passes through, and so if $\bar{k}$ is the number of obstacles π crosses, we have the upper bound $C_R \leq C_A + \bar{k}\epsilon/2$. Since $C_A \leq \kappa$, we have $C_R \leq \kappa + \bar{k}\epsilon/2$. We can

bound $\bar{k}$ by considering the following two cases. If $\kappa = C/\min_i c_i$, the minimum cost of an obstacle is 1, and so for each obstacle crossed, the path π incurs a cost of least $1 - \epsilon/2$. Therefore, $\bar{k} \leq \frac{\kappa}{1-\epsilon/2}$ and $C_R \leq \kappa + \frac{\kappa}{1-\epsilon/2} \cdot \epsilon/2 \leq \frac{1}{1-\epsilon/2}\kappa \leq (1+\epsilon)\kappa$. Otherwise, we have $\kappa = h$, which trivially implies $\bar{k} \leq \kappa$ since h is the total number of obstacles.

In conclusion, we have $C_R \leq (1+\epsilon)\kappa$, whose pre-scaled value is $\frac{(1+\epsilon)\kappa}{(\kappa/C)} = (1+\epsilon)C$, as claimed. Finally, the time complexity is dominated by an invocation of Dijkstra's algorithm on the graph G', which has $O(|V|\kappa/\epsilon)$ nodes and $O(|E|\kappa/\epsilon)$ edges. ■

If G is the viability graph constructed in this section then it always contains the shortest s–t path with cost at most C, i.e. $\alpha = 1$. Hence, by applying Theorem 20 to G we get a path of at most the optimum length and cost at most $(1+\epsilon)C$ in $\Omega(\frac{n^3}{\epsilon})$ time.

In the next section, we show that if we also allow an $(1+\epsilon)$ approximation of the path length, we can improve the running time by roughly an order of magnitude.

6.3 A Faster $(1+\epsilon)$-Approximation Algorithm

In this section, we describe our algorithm for sparsifying the graph $G = (V, E)$. We augment the graph by adding some vertices so that the number of viability edges can be sharply reduced, while approximately preserving the path lengths within the cost budget. Throughout the following discussion, we will respect the cost budget C, and only allow the path lengths to increase slightly. With that in mind, we use the notation $d_G(u, v)$ to denote the length of the shortest path in G from u to v whose cost is at most C. In this section we only use the definition of the cost of a path with respect to a viability graph. Recall that the cost of a path in a graph is the sum of the costs of the edges in the path.

Our sparse graph $H_\epsilon = (X_\epsilon, T_\epsilon)$ is defined for any $\epsilon > 0$, with $V \subseteq X_\epsilon$, and satisfies the following two conditions:

1. $d_G(u, v) \leq d_{H_\epsilon}(u, v) \leq (1+\epsilon)d_G(u, v)$ for all pairs $u, v \in V$.

2. The number of vertices and edges is $O(\frac{n}{\epsilon}\log n)$, that is, $|X_\epsilon|, |T_\epsilon| = O(\frac{n}{\epsilon}\log n)$.

We construct H_ϵ in two stages. In the first stage we construct a graph $H = (X, \Gamma)$ with $X \supseteq V$, $|X|, |\Gamma| = O(n \log n)$, and $d_G(u, v) \leq d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$. Next, we make $O(1/\epsilon)$ "copies" of H and combine them to construct H_ϵ. Once the graphs H and H_ϵ are constructed, we use the machinery of the previous section, namely Theorem 20, to efficiently find the approximately optimal shortest path within the cost budget.

Recall that all the obstacles in our input are convex, and therefore the shortest path in G does not cross the boundary of an obstacle more than twice. To avoid degenerate cases, we assume that all obstacle vertices are in general position, namely, no three vertices are collinear and all obstacles have non-zero area. We can, therefore, simplify the problem by replacing all the obstacles by their constituent boundary segments, where each obstacle vertex is assigned to its incident segment in the clockwise order. We now allocate the "obstacle removal" cost to these segments as follows: if c_i is the removal cost of obstacle i, then we allocate cost $c_i/2$ to each boundary segment of obstacle i. This ensures that any shortest path crossing the ith obstacle incurs a cost of c_i, while allowing us to reason about the geometry of just line segment obstacles.

We describe the construction of the sparse viability graph by explaining how to sparsify the "neighborhood" of an obstacle vertex, say, p. That is, we show which additional vertices are added and which viability edges are incident to p in the final sparse graph H. To simplify the discussion, we assume that p is at the origin, and we only discuss the edges incident to p that lie in the positive (north-east) quadrant; the remaining three quadrants are processed in the same way.

6.3.1 An $O(1)$-Approximation Algorithm

In this subsection we describe the construction of $H = (X, \Gamma)$ such that $|X|, |\Gamma| = O(n \log n)$, and $d_G(u, v) \leq d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$.

For a segment pq we use $\|pq\|_1$ to denote its L_1-length, i.e., $\|pq\|_1 = |x_p - x_q| + |y_p - y_q|$, where $p = (x_p, y_p)$ and $q = (x_q, y_q)$. For a polygonal path $\pi = p_0 p_1 \ldots p_k$, we use $\|\pi\|_1$ to denote its L_1-length, i.e., $\|\pi\|_1 = \sum_{i=1}^{k} \|p_{i-1} p_i\|_1$. We note that $\|\pi\|_1 \leq \sqrt{2}\|\pi\|$. We will construct a graph $H = (X, \Gamma)$ with the following property: For a pair of vertices $u, v \in V$ if G contains a path π from u to v of cost at most C, H contains a path $\bar{\pi}$ from u to v of cost at most C such that $\|\bar{\pi}\|_1 \leq \|\pi\|_1$. Hence $\|\bar{\pi}\| \leq \sqrt{2}\|\pi\|$ and thus $d_H(u, v) \leq \sqrt{2} d_G(u, v)$.

We are now ready to describe the algorithm for constructing H. It is a simple recursive algorithm and consists of the following steps:

1. Let x_m be the median x-coordinate of the points in V. We consider the vertical *split line* $\ell_v : x = x_m$ that partitions V into two almost equal-sized subsets V_l and V_r.

   (a) For each point $v \in V$, consider its projection $v' = (x_m, v_y)$ on the line ℓ_v. If $c(v, v') \leq C$, then add the *projection* vertex v' to X and the corresponding edge $e = (v, v')$ to Γ with length $\|vv'\|_1$ and cost $c(v, v')$.

   (b) Let s' be the first obstacle segment with positive slope that the projection segment vv' intersects. If s' intersects the split line ℓ_v, we add *bypass* vertices and edges to H as follows. Let v_1 be the point where vv' intersects s', and let v_2 be the point where s' intersects ℓ_v. We add bypass vertices v_1, v_2 on the segment s'. If v_2 lies above v_1, the bypass vertices are considered to be above the segment s', otherwise they are considered below the segment s'. (See also Figure 6.2.) We add the edges (v, v_1) and (v_1, v_2) to Γ with

lengths $\|vv_1\|_1, \|v_1v_2\|_1$ and costs $c(v, v_1), c(v_1, v_2)$, respectively. Observe that $c(v_1, v_2) = 0$.

(c) We repeat the procedure above for the first negative slope segment that vv' intersects.

(d) For two consecutive Steiner vertices w, w' (projection or bypass) on ℓ_v, if $c(w, w') \leq C$, then add the edge $e = (w, w')$ to Γ with length $\|ww'\|_1$ and cost $c(w, w')$.

(e) Recurse on the subsets V_l and V_r until $|V_l|, |V_r| \leq 1$.

2. Repeat the above process but this time using median y-coordinate y_m and a horizontal split line ℓ_h at $y = y_m$.

3. We add edges between consecutive vertices on the boundary of obstacles with cost 0.

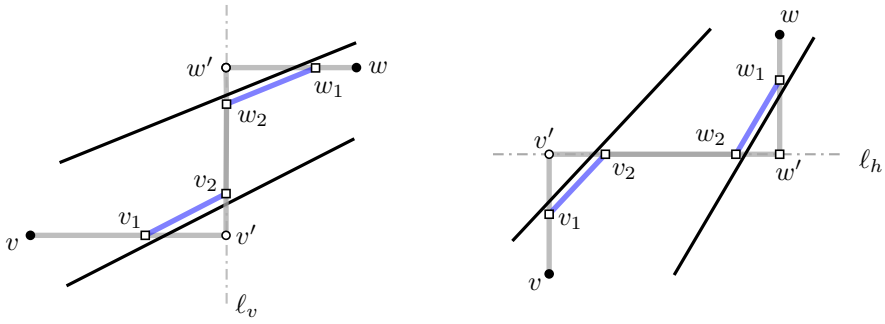


Figure 6.2: Steiner vertices due to vertical (left) and horizontal (right) split lines. Projections are shown with white dots, bypass vertices as squares, bypass edges shown in blue have cost zero.

At each recursive step of our algorithm, we need to find the first positive (negative) slope obstacle segment intersected by the projection segment vv', and compute the cost of all edges we add. In order to find the first positive (negative) slope segment say s', we can simply perform a point location query in $O(\log n)$ time [61] on positive (negative)

slope segments. If s', intersects both the projection segment vv' and the split line passing through v', we add the bypass vertices. For computing the edges costs, observe that bypass edges and the edges on the boundary of obstacles have both cost zero, and all other edges are either horizontal or vertical line segments, so we just need to compute the total cost of obstacle segments intersected by an axis aligned segment. We show how to do this for a horizontal projection segment vv' and all other cases follow similarly. We preprocess all the obstacle segments in a segment tree based data structure S. Using fractional cascading and increasing the fan-out of the segment tree [62, 61], a (weighted) counting query runs in $O(\log n)$ time. During each recursive call, we simply query S to compute the cost of the segment vv'. However, we need to be careful in including the cost of the obstacle segment that v lies on. More precisely, if P_i is the obstacle incident to v, we include the cost $c_i/2$ to the cost of segment vv' only if vv' intersects the interior of P_i (which we can decide in constant time).

We can easily obtain the following lemma.

Lemma 57 *Every input vertex adds Steiner vertices on $O(\log n)$ split lines. Moreover, graph H has size $O(n \log n)$ and can be constructed in $O(n \log^2 n)$ time.*

Proof: In each recursive call with n vertices, the total number of new projection vertices added is at most n (one per projection). For each projection we can have at most two bypass vertices, so the number of bypass vertices is $O(n)$. The number of edges is also $O(n)$. Combined over all recursive calls, it is easy to observe that each input vertex adds Steiner vertices on $O(\log n)$ split lines and the total number of vertices and edges in H is $O(n \log n)$.

For the running time – at each recursive level we make a total of $O(n)$ queries to find the bypass vertices and the edge costs. These take $O(n \log n)$ time in total giving us a recurrence that solves to $O(n \log^2 n)$. ■

It is important to note here that a similar recursive algorithm was first used by Clarkson et al. [60] to compute L_1 shortest obstacle-avoiding paths in the plane – each vertex was projected on $O(\log n)$ split lines and on the obstacle closest to it in all four directions. This was enough to capture obstacle-avoiding shortest paths (as they lie entirely in free space) but since obstacle-removing shortest paths can also go through obstacles, things get quite complicated. In particular, it is not clear that which of the $O(n)$ nearby obstacles (in each direction) should a vertex be projected on. We address this challenge in Step 1b of our algorithm by adding bypass vertices. Since we need to guarantee that the sparsification preserves the L_1 length as well as the cost of the shortest path, our correctness argument is quite different and can be viewed as a more general form of the result by [60].

6.3.2 Proof of Correctness

We now prove that $d_H(u, v) \leq \sqrt{2} d_G(u, v)$ for all $u, v \in V$. More precisely, if we set the length of each edge $e = (u, v)$ in G to be $\|uv\|_1$, then we show that $d_H(u, v) \leq d_G(u, v)$. We basically show that for any edge $e = (u, v)$ in G there is a path π_e from u to v in H such that $c(\pi_e) \leq c(u, v)$ and $\|\pi_e\|_1 \leq \|uv\|_1$. This claim is established in Lemma 60, whose proof relies on the following Lemmas 58 and 59.

For convenience, we introduce the notion of the region R_{pq} defined by two obstacle vertices $p, q \in V$. Let $\bar{R}_{pq}$ be the rectangle with p and q as lower left and upper right corners respectively. Now, let s_x (resp. s_y) be the first obstacle segment of positive slope that intersects the two sides of $\bar{R}_{pq}$ below (resp. above) the diagonal pq. We define $R_{pq} = \bar{R}_{pq} \setminus (\mathcal{B}(s_x) \cup \mathcal{A}(s_y))$, where $\mathcal{B}(s_x)$ is the area below segment s_x, and $\mathcal{A}(s_y)$ is the area above s_y. If a segment s_x or s_y does not exist then $\mathcal{B}(s_x) = \emptyset$ and $\mathcal{A}(s_y) = \emptyset$. (See also Figure 6.3.)

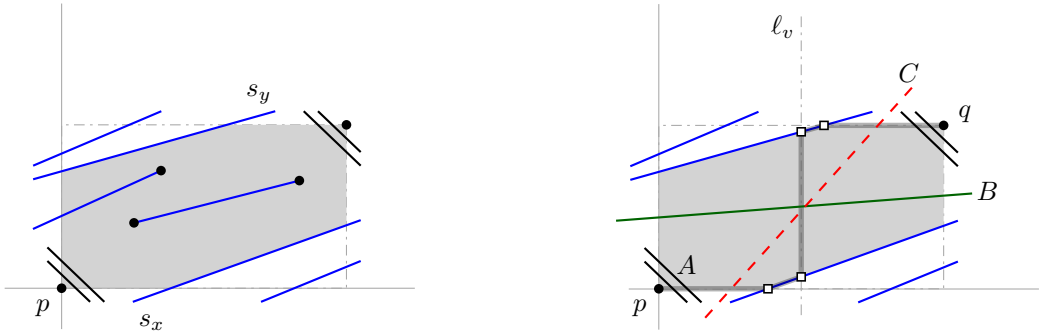


Figure 6.3: The region R_{pq} is shown shaded. If R_{pq} does not contain obstacle vertices, the type A, B, C obstacle segments that may intersect R_{pq} are shown on the right. Observe that type B and type C segments cannot both exist in R_{pq}.

Lemma 58 *Let (p, q) be an edge in G with cost $c(p, q)$. If the region R_{pq} does not contain an obstacle vertex, then there exists a path π_{pq} in H that is entirely contained in R_{pq} such that $\|\pi_{pq}\|_1 = \|pq\|_1$ and $c(\pi_{pq}) = c(p, q)$.*

Proof: Since R_{pq} does not contain any obstacle vertex there are only three types of obstacle segments that intersect R_{pq}. (See also Figure 6.3.)

1. *Type A* : these obstacle segments have negative slope and intersect both vertical and horizontal segments of R_{pq} adjacent to either p or q.

2. *Type B* : obstacle segments that intersect both vertical segments of R_{pq}.

3. *Type C* : obstacle segments that intersect both horizontal segments of R_{pq}.

It is easy to see that segments of type B and C cannot both exist in R_{pq} because the obstacle segments are non-intersecting. From the construction of H there is always a vertical and a horizontal split line between two obstacle vertices. Let ℓ_v (ℓ_h) be the first vertical (horizontal) split line in the recursion that we consider between the vertices p, q. There are three cases.

- *Only Type A segments exist in R_{pq}.* This case is taken care by the Steiner vertices on the vertical (or horizontal) split line ℓ_v. More precisely, ℓ_v may intersect both s_x

and s_y, one of them, or even neither of them. We show what happens in the case where ℓ_v intersects both s_x and s_y and the other cases follow easily. Since there are no obstacle vertices in R_{pq}, s_x, s_y are the first positive slope segments intersected by the projections of p, q on ℓ_v. So we have created bypass vertices p_1, p_2 and q_1, q_2 on s_x, s_y. The path π_{pq} is defined as $\pi_{pq} = pp_1p_2q_2q_1q$ and it is easy to see that $\|\pi_{pq}\|_1 = \|pq\|_1$. Moreover, both π_{pq} and the edge pq cross one time the same set of obstacle segments (only type A), so we have that $c(\pi_{pq}) = c(p, q)$.

- *Type B segments exist in R_{pq}.* In this case, observe that type B edges do not intersect with the horizontal projection segments adjacent to p and q on the vertical split line, and therefore we can use the exact same path π_{pq} as the previous case. The cost of the type B segments needs to be included but since the edge pq must cross these segments, we have that $c(\pi_{pq}) = c(p, q)$.

- *Type C segments exist in R_{pq}.* This case is symmetric to the previous case using the *horizontal* split line ℓ_h.

■

Lemma 59 *Let (p, q) be an edge in G with cost $c(p, q)$. If the region R_{pq} contains one or more obstacle vertices, then there exists an obstacle vertex $r \in R_{pq}$ such that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$, and $c(p, r) + c(r, q) \leq c(p, q)$.*

Proof: We prove the lemma by exhibiting a vertex r such that (i) the triangle Δprq does not contain any other obstacle vertex, and (ii) no obstacles segment intersects the interiors of both pr and rq. Such a choice of r suffices for our proof since $r \in R_{pq}$ implies that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and we get $c(p, r) + c(r, q) \leq c(p, q)$ because any obstacle segment crossing either pr or rq must also cross pq, otherwise that obstacle segment would terminate inside the triangle which contradicts the choice of r. Next, we show

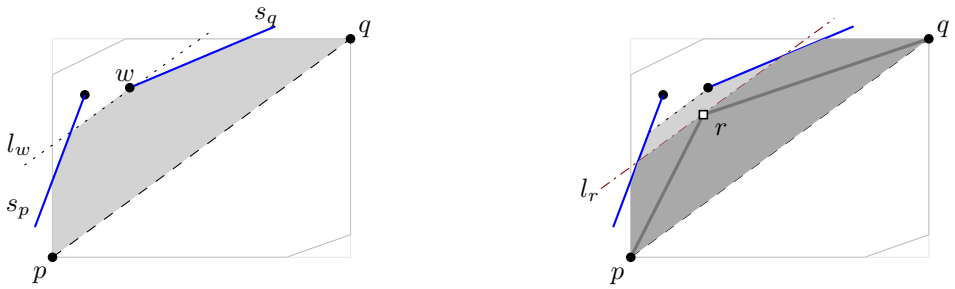


Figure 6.4: The region T_{pq} is shown shaded on left. If $r \in T_{pq}$ is the vertex closest to pq, then the region $T'_{pq} \subseteq T_{pq}$(shown shaded in dark on right) cannot contain an obstacle vertex.

how to find such a vertex. We restrict our search for this vertex r in a convex polygon $T_{pq} \subseteq R_{pq}$ which we construct in the following way. (See also Figure 6.4.) Observe that the diagonal $\overline{pq}$ divides the region R_{pq} into two subsets – one above and one below it. We consider the subset R'_{pq} that contains at least one obstacle vertex. Since, R_{pq} contains at least one obstacle vertex, such a subset always exists. Without loss of generality, we can assume that R'_{pq} lies above $\overline{pq}$. Now, let S_{pq} be the set of all obstacle segments that intersect a vertical or a horizontal segment of the boundary $\partial R'_{pq}$, and let $s_p, s_q \in S_{pq}$ be the segments that intersect $\partial R'_{pq}$ closest to p and q respectively. From the endpoints of s_p, s_q that lie in R'_{pq}, let w be the endpoint closest to the segment $\overline{pq}$. Moreover, let l_w be the line parallel to $\overline{pq}$ that passes through w. Now we simply clip off the region of R'_{pq} that lies above l_w. More precisely, this gives us the quadrilateral $R''_{pq} = R'_{pq} \setminus \mathcal{A}(l_w)$, where we use $\mathcal{A}(s)$ for the region above segment s. Finally, we define the convex polygon $T_{pq} = R''_{pq} \setminus (\mathcal{A}(s'_p) \cup \mathcal{A}(s'_q))$, where s'_p, s'_q are the subsegements of s_p, s_q respectively that lie inside the quadrilateral R''_{pq}.

From the set of obstacle vertices that lie *inside or on the boundary* of T_{pq}, we choose the vertex r to be the one that minimizes the area of the triangle Δprq, or equivalently, be the one that has the minimum distance from the segment pq. Observe that the boundary of region T_{pq} contains the obstacle vertex w, so we will always find one such r. It is easy

to see that the triangle Δprq is a subset of T_{pq} and does not contain an obstacle vertex or else it would not have the minimum area. It remains to show that there cannot be an obstacle segment that crosses both pr and rq. To this end, let l_r be a line parallel to pq passing through r. Observe that the region $T'_{pq} = T_{pq} \setminus \mathcal{A}(l_r)$, i.e., the region in T_{pq} that lies below l_r, cannot contain an obstacle vertex by the choice of r. So any obstacle segment s_j that crosses both pr and rq must intersect $\partial R'_{pq}$ at either the vertical segment between p and s_p or the horizontal segment between s_q and q which is a contradiction. (See also Figure 6.4.) ■

Finally, we prove the main result of this section.

Lemma 60 *Let (p, q) be an edge in G with cost $c(p, q)$. There is a path $\pi_{pq} \in H$ such that $\|\pi_{pq}\|_1 = \|pq\|_1$ and $c(\pi_{pq}) \leq c(p, q)$. Moreover, the path π_{pq} lies in the region R_{pq}.*

Proof: We prove this by induction on the number of obstacle vertices in the region R_{pq}. Our base case is when the region R_{pq} does not contain an obstacle vertex. Applying Lemma 58 gives us the desired path π_{pq} in H. For the inductive step, let j be the number of obstacle vertices in the region R_{pq} and assume that the lemma holds for all edges (u, v) such that the region R_{uv} contains $i < j$ obstacle vertices. Using Lemma 59 we find an intermediate vertex r such that $\|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and $c(p, r) + c(r, q) \leq c(p, q)$. This gives us two disjoint sub-regions $R_{pr} \subset R_{pq}$ and $R_{rq} \subset R_{pq}$ each with at least one less obstacle vertex than the region R_{pq}. By our induction hypothesis, we get the disjoint subpaths π_{pr} from p to r and π_{rq} from r to q in H. We then join these two paths at vertex r to obtain path π_{pq} that lies within the region R_{pq}. Moreover, we have that $\|\pi_{pq}\|_1 = \|\pi_{pr}\|_1 + \|\pi_{rq}\|_1 = \|pr\|_1 + \|rq\|_1 = \|pq\|_1$ and $c(\pi_{pq}) = c(\pi_{pr}) + c(\pi_{rq}) \leq c(p, r) + c(r, q) \leq c(p, q)$. ■

6.3.3 An $(1+\epsilon)$-Approximation Algorithm

We now describe how to use the preceding construction to define our final sparse graph H_ϵ. A direction in $\mathbb{R}^2$ can be represented as a unit vector $\mathbf{u} \in \mathbb{S}^1$. Let $\mathsf{N} \subset \mathbb{S}^1$ be a set of $O(1/\epsilon)$ unit vectors such that the angle between two consecutive points of N is at most ϵ. For each $\mathbf{u} \in \mathsf{N}$, we construct a graph $H^{\mathbf{u}}$ by running the algorithm in Section 6.3.1 but regarding $\mathbf{u}$ to be the y axis — i.e., by rotating the plane so that $\mathbf{u}$ becomes parallel to the y-axis and measure L_1-distance in the rotated plane. Set $H_\epsilon = \bigcup_{\mathbf{u}\in\mathsf{N}} H^{\mathbf{u}}$. Notice that the number of vertices and edges in H_ϵ is $O(\frac{n}{\epsilon}\log n)$. The following lemma follows easily by the discussion above.

Lemma 61 *For any pair $u, v \in V$, we have that $d_{H_\epsilon}(u,v) \leq (1+\epsilon)d_G(u,v)$.*

Proof: Let (p,q) be an edge of the path π_{uv} from u to v in graph G with $c(p,q) \leq C$, and let $\mathbf{u}$ be a direction in N such that the angle between the segment pq and $\mathbf{u}$ is at most ϵ. From the definition of N there is always such a direction $\mathbf{u}$. From Lemma 60 there is a path π from p to q in $H^{\mathbf{u}}$ such that $c(\pi) \leq c(p,q)$ and $\|\pi\|_1 = \|pq\|_1$. Recall that the angle between $\mathbf{u}$ and pq is at most ϵ, so $\|pq\|_1 \leq (\cos\epsilon + \sin\epsilon)\|pq\|$. We conclude that $\|\pi\| \leq (\cos\epsilon + \sin\epsilon)\|pq\| \leq (1+\epsilon)\|pq\|$, for $\epsilon \in (0,1)$. The result follows. ■

From the above lemma, it follows that the graph H_ϵ preserves pairwise shortest path distances within a factor of $(1+\epsilon)$ and at most the same cost with graph G. Let L^* be the length of the shortest s–t path in the plane that has cost at most C. Since there exists a s–t path of length at most L^* and cost at most C in the viability graph G, there exists a s–t path in H_ϵ of length $(1+\epsilon)L^*$ and the same cost. Applying Theorem 20 with $\alpha = (1+\epsilon)$ on H_ϵ gives the following result.

Theorem 21 *Let P be a set of h convex polygonal obstacles with n vertices, s, t be two obstacle vertices and $C \in \mathbb{R}$ be a parameter. If L^* is the length of the shortest s–t path*

with cost at most C, a s–t path with length at most $(1+\epsilon)L^*$ and cost at most $(1+\epsilon)C$ can be computed in $O(\frac{nh}{\epsilon^2}\log n \log \frac{n}{\epsilon})$ time.

6.4 Shortest Path Queries

We now describe a near-linear space data structure to answer approximate distance queries from a fixed obstacle vertex s subject to the obstacle removal budget in $O(\frac{1}{\epsilon}\log^2 n)$ time. The data structure is then extended to handle two-point shortest path queries in $O(\frac{1}{\epsilon^2}\log^2 n)$ time with near-quadratic space.

The key idea relies on the following observation. Without loss of generality, assume that the points s and t lie in the exterior of all obstacles and let us also assume that s, t were part of the input. Now consider the shortest s–t path in the graph H_ϵ and let t' be the vertex preceding t in this path. It is easy to see that t' must be a Steiner vertex (projection or bypass) as there are no direct edges in H_ϵ between two input vertices that do not lie on the same obstacle. All such edges must cross some split line at Steiner vertices. Therefore, the last edge (t', t) in the path is the segment obtained by projecting t on some split line ℓ. Now, suppose we have precomputed the paths to all Steiner vertices on all split lines, then we can find the shortest path to t by simply finding the neighbor of t' on ℓ. Using Lemma 57, we know that t can be projected on $O(\frac{1}{\epsilon}\log n)$ split lines, which gives $O(\frac{1}{\epsilon}\log n)$ choices for ℓ.

6.4.1 Preprocessing

We apply the algorithm preceding Theorem 20 on the graph H_ϵ that we constructed in the previous section. More precisely, first we multiply the cost of all obstacles by h/C so that the target cost becomes h. Next we create an auxiliary graph H'_ϵ with $O(\frac{h}{\epsilon})$ copies of each vertex in H_ϵ. Running Dijkstra's algorithm on H'_ϵ with source s computes a shortest

path to each vertex in H'_ϵ. Now for each vertex v in H_ϵ, we maintain arrays $dist_v, pred_v$ each with size $1 + \frac{h}{\epsilon} = O(\frac{h}{\epsilon})$. We store the length of the shortest path found by Dijkstra's algorithm from s to $v_{i\epsilon}$ (i-th copy of vertex v) at $dist_v(i)$ and its predecessor in $pred_v(i)$. In addition, for each direction $\mathbf{u} \in \mathsf{N}$ that we defined in the previous section we maintain two data structures:

- A segment tree [62] based data structure $S_\mathbf{u}$ that we also used in Section 6.3.1 to compute the cost of an axis aligned segment in $O(\log n)$ time.
- A balanced search tree $T_\mathbf{u}$ over all the vertical (resp. horizontal) split lines, which is basically the recursion tree corresponding to the algorithm from Section 6.3.1. More precisely, the root of $T_\mathbf{u}$ is the split line ℓ_m (at the median x-coordinate x_m), and the left and right children are the split lines added during recursive processing of points to the left and right of ℓ_m respectively.

Moreover, for every split line ℓ, we maintain a search tree over all the Steiner vertices that lie on ℓ. Overall, our data structure consists of all arrays $dist_v, pred_v$, $O(\frac{n}{\epsilon})$ search trees, and $O(\frac{1}{\epsilon})$ segment trees $S_\mathbf{u}$. The size of the data structure is $O(\frac{nh}{\epsilon^2} \log n)$ and the preprocessing time is $O(\frac{nh}{\epsilon^2} \log n \log \frac{n}{\epsilon})$.

6.4.2 Query

The query procedure consists of two parts. Given the target query point t, we first find a subset of $O(\frac{1}{\epsilon} \log n)$ split lines L that we need to search. Next, for each line $\ell \in L$, we find the Steiner vertex t' created by projecting t on ℓ and then find the path to t using one of the two neighbors of t' on ℓ. Let v denote a neighbor of t' on ℓ. Finally, we take the shortest of all $O(\frac{1}{\epsilon} \log n)$ candidate paths.

In order to find the subset of split lines we use the search tree $T_\mathbf{u}$ over the set of all split lines for a direction $\mathbf{u} \in \mathsf{N}$. For a node $z \in T_\mathbf{u}$, if t lies in the region left of split line

at z we search the left child, else we search the right child. Searching $T_{\mathbf{u}}$ in this way, we reach a leaf node such that the associated region contains exactly one obstacle vertex w and the query point t. In this case we add a new split line ℓ^* between w and t and add Steiner vertices for the obstacle vertex w on ℓ^*. This gives us a total of $O(\log n) + 1$ split lines per direction that we need to search, and in total $O(\frac{1}{\epsilon} \log n)$ split lines.

To compute the candidate paths for a given a split line ℓ, we consider the Steiner vertices – projection t' and bypass t_1, t_2 – for the query point t. The shortest path from ℓ to t may either be $t' \to t$ or $t_2 \to t_1 \to t$. We find a neighbor v of t' or t_2 on ℓ (at most two neighbors are possible). We now consider the section of the path π_{vt} from v to t. If the arrays $dist_v, pred_v$ are not precomputed, which can happen if v is the projection of an obstacle vertex w on the new split line ℓ^*, we set $v = w$ and include the path from w to t along the split line ℓ^* to π_{vt}. (See also Figure 6.5.) At this point we have found

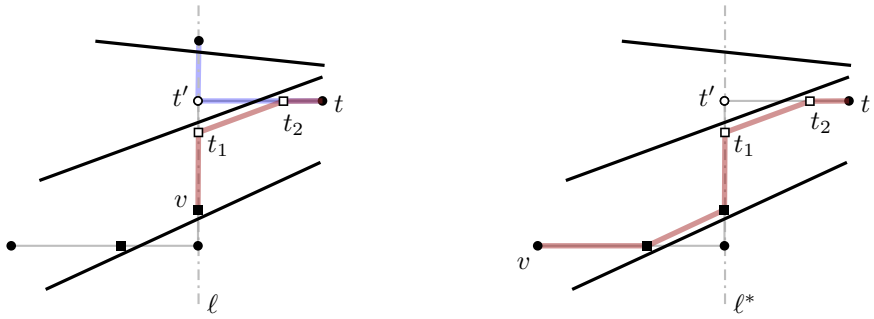


Figure 6.5: Computing path from a query point t to one of the vertices in H_ϵ – using a split line that already exists in H_ϵ (left) and using a new split line ℓ^* added at query time (right). The suffix path π_{vt} is shown shaded in red.

a vertex v such that $dist_v, pred_v$ are precomputed for all cost values $0, \epsilon, 2\epsilon, \ldots, h$. Since the cost of bypass edges is zero, and all other segments in the path π_{vt} are axis-aligned, we can compute the cost $c(\pi_{vt})$ using the segment tree $S_{\mathbf{u}}$. The remaining cost budget is $h - c(\pi_{vt})$ which we *round up* for looking up in the $dist_v, pred_v$ arrays. More precisely, let j be the integer such that $(j-1)\epsilon \leq h - c(\pi_{vt}) \leq j\epsilon$, then we compute the length of the candidate s–t path via v as $dist_v(j) + \|\pi_{vt}\|_1$. Finding the vertex v for a given split line

takes $O(\log n)$ time, and compute the index $j = \left\lceil \frac{h - c(\pi_{vt})}{\epsilon} \right\rceil$ in constant time. Therefore, computing the value $dist_v(j)$ takes $O(\log n)$ time in total. Finally, we take the minimum over all $O(\frac{1}{\epsilon} \log n)$ choices of v (one per split line) to obtain the shortest path π_{st} using the *pred* arrays. Using a similar argument as in the proof of Theorem 20, one can show that the length of π_{st} is at most $(1+\epsilon)$ times optimal and the cost is $(1+\epsilon)C$. The total query time is $O(\frac{1}{\epsilon} \log n \cdot \log n) = O(\frac{1}{\epsilon} \log^2 n)$.

Improving Space Complexity Instead of computing the path itself, one may ask to just find the length of the shortest s–t path of cost at most C for some query point t. We can answer such queries approximately in $O(\frac{1}{\epsilon} \log^2 n)$ time using $O(\frac{n}{\epsilon^2} \log n \log \frac{h}{\epsilon})$ space which improves on the $O(\frac{nh}{\epsilon} \log n \log \frac{h}{\epsilon})$ space requirement for finding the exact path. The main idea is that instead of storing $O(\frac{h}{\epsilon})$ distance values in $dist_v$ for cost $0, \epsilon, 2\epsilon, \ldots, \frac{h}{\epsilon}\epsilon$, we store a subset of $O(\frac{1}{\epsilon} \log \frac{h}{\epsilon})$ values. More precisely, we only store the distance values corresponding to the cost $j\epsilon$ where j is an integer such that $(j-1)\epsilon < \epsilon(1+\epsilon)^i \leq j\epsilon$, for all i in $0, 1, 2, \ldots, \log_{1+\epsilon} \frac{h}{\epsilon}$. The size of $dist_v$ arrays for each vertex v is therefore $O(\log_{1+\epsilon} \frac{h}{\epsilon}) = O(\frac{1}{\epsilon} \log \frac{h}{\epsilon})$. Note that the reason this works for the path length (but not for exact path) is because we only need to find the penultimate vertex in the path, so an accurate linking of predecessors using $pred_v$ arrays is not needed.

We will now adapt the query procedure as follows. First, we compute the vertex v as usual in $O(\log n)$ time. The only change is in how we find an appropriate index j in the 'now sparse' array $dist_v$. Once that is done, we return the length of path via v as $dist_v(j) + \|\pi_{vt}\|_1$ as usual. Let i be an integer such that $\epsilon(1+\epsilon)^{i-1} < h - c(\pi_{vt}) \leq \epsilon(1+\epsilon)^i$. We use the index j stored in $dist_v$ corresponding to this value of i. The path π_{st} is taken to be minimum over all choices of v. We can now prove the following bound on the cost of π_{st}.

Lemma 62 *The cost of π_{st} is at most $(1+5\epsilon)C$.*

Proof: We first apply the scaling factor h/C to the costs and show that $c(\pi_{st}) \leq (1+5\epsilon)h$. Since we try all split lines, without loss of generality, assume that v is the vertex on the split line that gives us the path π_{st}. From Theorem 20, $dist_v(j)$ stores the length of a path in the plane with cost at most $j\epsilon+h\epsilon$. Here j is such that $(j-1)\epsilon < \epsilon(1+\epsilon)^i \leq j\epsilon$. Hence, the cost of the corresponding path from s to v is at most $j\epsilon+h\epsilon \leq \epsilon(1+\epsilon)^i+\epsilon+h\epsilon$. If $i \geq 1$, we bound the cost of π_{st} as follows. For the next inequalities, we note that $\epsilon(1+\epsilon)^{i-1} + c(\pi_{vt}) < h$.

$$\begin{aligned} c(\pi_{st}) &\leq \epsilon(1+\epsilon)^i + h\epsilon + \epsilon + c(\pi_{vt}) \\ &\leq (1+\epsilon)\left(\epsilon(1+\epsilon)^{i-1} + h\epsilon + \epsilon + c(\pi_{vt})\right) \\ &\leq (1+\epsilon)(h+h\epsilon+\epsilon) \\ &\leq (1+\epsilon)(1+2\epsilon)h \\ &\leq (1+5\epsilon)h \end{aligned}$$

If $i = 0$, then $c(\pi_{st}) \leq h\epsilon + \epsilon + c(\pi_{vt}) \leq (1+2\epsilon)h$. By scaling back the costs of the obstacles we get $c(\pi_{st}) \leq (1+5\epsilon)C$. ■

Given a split line, we can find the vertex v in $O(\log n)$ time and find index j in additional $O(\log n)$ time (assuming the sparse $dist_v$ arrays are sorted by the index j). Therefore, query time is $O(\log n)$ per split line and $O(\frac{1}{\epsilon}\log^2 n)$ in total. Constructing the data structure for $\epsilon \leftarrow \epsilon/5$, we obtain the following theorem.

Theorem 22 *Let* P *be a set of h convex polygonal obstacles with n vertices, s be an obstacle vertex, and $C \in \mathbb{R}$ be a parameter. A data structure of $O(\frac{nh}{\epsilon^2}\log n)$ size can be constructed in $O(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon})$ time such that, given a query point $t \in \mathbb{R}^2$, a path π_{st} can be returned with cost $(1+\epsilon)C$ and length at most $(1+\epsilon)$ times the optimal in $O\big(\frac{1}{\epsilon}\log^2 n + k_{st}\big)$ time, where k_{st} is the number of edges of π_{st}. The length of the path π_{st}*

can be returned in time $O\big(\frac{1}{\epsilon}\log^2 n\big)$ *using a data structure of size* $O(\frac{n}{\epsilon^2}\log n \log\frac{h}{\epsilon})$.

Two point queries Now we briefly explain how to extend the above data structure to handle two point queries. That is, both s, t are part of the query. During the preprocessing, we store distance values $dist_{uv}$ (similarly $pred_{uv}$) for every pair of vertices u, v in H_ϵ for all cost values $0, \epsilon, 2\epsilon, \ldots, h$. The idea now is to find the neighbor u of s on some split line ℓ_s and neighbor v of t on split line ℓ_t. We compute the cost of paths π_{sv} and π_{vt} as before and set the length of this candidate s–t path to be $dist_{uv}(j) + \|\pi_{su}\|_1 + \|\pi_{vt}\|_1$. Here j is the smallest integer such that $h - c(\pi_{su}) - c(\pi_{vt}) \leq j\epsilon$. We take the minimum across $O(\frac{1}{\epsilon^2}\log^2 n)$ choices of u and v.

Theorem 23 *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices, and* $C \in \mathbb{R}$ *be a parameter. A data structure of* $O(\frac{n^2h}{\epsilon^3}\log^2 n)$ *size can be constructed in* $O(\frac{n^2h}{\epsilon^3}\log^2 n \log\frac{n}{\epsilon})$ *time such that, given two query points* $s, t \in \mathbb{R}^2$*, a path* π_{st} *can be returned with cost at most* $(1+\epsilon)C$ *and length at most* $(1+\epsilon)$ *times the optimal in* $O(\frac{1}{\epsilon^2}\log^2 n + k_{st})$ *time, where* k_{st} *is the number of edges of* π_{st}*. The length of the path* π_{st} *can be returned in* $O(\frac{1}{\epsilon^2}\log^2 n)$ *time using a data structure of size* $O(\frac{n^2}{\epsilon^3}\log^2 n \log\frac{h}{\epsilon})$.

6.5 Stochastic Shortest Path

In this section, we consider a stochastic model of obstacles where the existence of each obstacle $P_i \in \mathsf{P}$ is an independent event with known probability β_i. That is, P_i is part of the input with probability β_i and is not part of the input with probability $1 - \beta_i$. We define the probability of path π_{st} as $\prod_{P_i \in S}(1-\beta_i)$ where $S \subseteq \mathsf{P}$ is the set of obstacles that this path goes through (assuming they did not exist). In such a setting, our goal is to compute the approximate shortest path that has probability more than a given threshold $\beta \in (e^{-1}, 1]$.

Let L_β denote the length of the shortest path from s to t with probability at least β. We convert the multiplicative costs to additive costs by setting $c_i = -\ln(1-\beta_i)$ for each obstacle and setting $C = -\ln\beta$. Using Theorem 21, we find a path π_{st} with length $L(\pi_{st}) \leq (1+\epsilon)L_\beta$ and cost $c(\pi_{st}) \leq (1+\epsilon)C$. It can be shown that π_{st} has probability at least $(1-\epsilon)\beta$.

Theorem 24 *Let* P *be a set of* h *convex polygonal obstacles with* n *vertices, where each obstacle* $P_i \in \mathsf{P}$ *exists independently with a probability* β_i*,* s, t *be two obstacle vertices and* $\beta \in (e^{-1}, 1]$ *be a parameter. If* L_β *is the length of the shortest* s*–*t *path with probability at least* β*, a* s*–*t *path with length at most* $(1+\epsilon)L_\beta$ *and probability at least* $(1-\epsilon)\beta$ *can be computed in* $O(\frac{nh}{\epsilon^2}\log n \log\frac{n}{\epsilon})$ *time.*

Most likely path We now consider the following question – given a bound L on the length of the path, what is the s–t path with maximum probability? We need a bound on the path length or else there is always a path of probability 1. To answer this question, we can again take negative logarithms of probabilities to transform into an additive cost model and construct the graph H_ϵ as before. Now instead of applying Theorem 20 on H_ϵ, we construct a new graph H_ϵ^* that is exactly the same as H_ϵ, but with length and cost parameters on edges interchanged. More precisely, for an edge $e \in H_\epsilon$ with length l_e and cost c_e, we have an edge $e^* \in H_\epsilon^*$ with length c_e and cost l_e. Next we apply Theorem 20 on the graph H_ϵ^* with $C = (1+\epsilon)L$, and scale all costs with a parameter $O(\frac{n}{C\epsilon}\log n)$, such that the target cost is scaled to $O(\frac{n}{\epsilon}\log n)$. We choose this value because a shortest path in H_ϵ can have $O(\frac{n}{\epsilon}\log n)$ edges. This gives us the following result.

Theorem 25 *Let* P *be a set of* h *convex obstacles with* n *vertices,* s, t *be two obstacle vertices, and* $L \in \mathbb{R}$ *be a parameter. If* β_M *is the maximum probability of a path from* s *to* t *with length at most* L*, a path* π_{st} *with length at most* $(1+\epsilon)L$ *and probability at least* β_M *can be computed in* $O(\frac{n^2}{\epsilon^3}\log^2 n \log\frac{n}{\epsilon})$ *time.*

6.6 Bibliographic Notes

The cost-version of obstacle removing shortest path problem is closely related to the problem of computing a shortest path through weighted polygonal regions [63, 64, 46] where the length of a path is defined as the weighted sum of Euclidean or L_1 lengths of the subpaths within each region. However, in our setting there is only a one-time fixed cost for passing through a region, and therefore does not depend on the length of the subpath that lies inside the region.

The stochastic formulation of our problem is also related to some shortest path problems under uncertainty [65, 66, 67, 68]. However, these results assume existence of a graph whose edges have either an existence probability or a distribution over their lengths. In contrast, our definition is purely geometric where the existence of obstacles is an uncertain event. Our problem can also be seen as a variant of geometric bi-criteria shortest path problem [69, 70, 71, 72, 73], as our objective is to compute the shortest path with a constraint on the total cost of obstacles that we remove.

As for the queries, the shortest path map [30] has linear size and can answer Euclidean shortest path queries with a fixed source in $O(\log n)$ time. If both s, t are part of the query, quadratic space data structures [74, 75] exist for L_1 shortest path queries and super quadratic space data structures [76] for L_2 shortest path queries. Similar results exist for rectilinear shortest path queries among disjoint weighted rectilinear and convex obstacles [74, 77], and for bi-criteria shortest path problems [70, 71, 73].

Chapter 7

The Maximum Exposure Problem

Let $S = (P, \mathcal{R})$ be a geometric set system, also called a *range space*, where P is a set of points and each $R \in \mathcal{R}$ is a subset of P, also called a range. We say that a point $p \in P$ is *exposed* if no range in $\mathcal{R}$ contains p. The *max-exposure* problem is defined as follows: given a range space $(P, \mathcal{R})$ and an integer parameter $k \geq 1$, remove k ranges from $\mathcal{R}$ so that a maximum number of points are exposed. That is, we want to find a subfamily $\mathcal{R}^* \subseteq \mathcal{R}$ with $|\mathcal{R}^*| = k$, so that the number of exposed points in the (reduced) range space $(P, \mathcal{R} \setminus \mathcal{R}^*)$ is maximized. In this chapter, we design some approximation algorithms as well as some hardness of approximation lowerbounds for max-exposure. Although, We are primarily interested in range spaces defined by a set of points in two dimensions and ranges defined by axis-aligned rectangles, some of these results are also applicable to any polygonal range with constant number of sides and pseudodisks.

The max-exposure problem arises naturally in many geometric coverage settings. For instance, if points are the location of clients in the two-dimensional plane, and ranges correspond to coverage areas of facilities, then exposed points are those not covered by any facility. The max-exposure problem in this case gives a worst-case bound on the number of clients that can be exposed if an adversary disables k facilities. Similarly, in

distributed sensor networks, ranges correspond to *sensing zones*, points correspond to physical assets being monitored by the network, and the max-exposure problem computes the number of assets exposed when k sensors are compromised.

More broadly, the max-exposure problem is related to the densest k-subgraph problem in *hypergraphs*. In the *densest k-subhypergraph* problem, we are given a hypergraph $H = (X, E)$, and we want to find a set of k vertices with a maximum number of induced hyperedges. In general hypergraphs, finding k-densest subgraphs is known to be (conditionally) hard to approximate within a factor of $n^{1-\epsilon}$, where n is the number of vertices. The max-exposure problem is equivalent to the densest k-subhypergraph problem on a *dual* hypergraph, whose vertices X corresponds to the ranges $\mathcal{R}$, and whose hyperedges correspond to the set of points P. Specifically, each point $p \in P$ corresponds to a hyperedge adjacent to the set of ranges containing the point p. In the rest of this chapter, we will use $n = |\mathcal{R}|$ for the number of ranges in $\mathcal{R}$ and $m = |P|$ for the number of points. We show that if the range space is defined by *convex polygons*, then the max-exposure problem is just as hard as the densest k-subhypergraph problem. However, for ranges defined by *axis-aligned rectangles*, one can achieve better approximation.

Summary of Results and Techniques

We will now briefly mention all the results that we discuss in this chapter and the techniques we apply to achieve these results.

1. We show that the max-exposure problem is NP-hard and assuming the *dense vs random* conjecture, it is also hard to approximate better than a factor of $O(n^{1/4})$ even if the range space is defined by *only two types of rectangles* in the plane. For range space defined by convex polygons, we show that max-exposure is equivalent to densest k-subhypergraph problem, which is hard to approximate within a factor

of $O(n^{1-\epsilon})$.

2. When ranges are defined by translates of a *single* rectangle, we give a polynomial-time approximation scheme (PTAS) for max-exposure. The PTAS stands in sharp contrast to the inapproximability of ranges defined by *two* types of rectangles. Moreover, as an easy consequence of this result, we obtain a constant approximation when the ratio of longest and smallest side of rectangles in $\mathcal{R}$ is bounded by a constant. However, we do not know if max-exposure with translates of a single rectangle can be solved in polynomial time or is NP-hard.

3. For ranges defined by arbitrary rectangles, we present a simple greedy algorithm that achieves a bicriteria $O(k)$-approximation. No such approximation is possible for general hypergraphs. If rectangles in $\mathcal{R}$ have a bounded aspect ratio, the approximation improves to $O(\sqrt{k})$. For pseudodisks with *bounded-ply* (no point in the plane is contained in more than a constant number of disks), this algorithm achieves a constant approximation.

The PTAS is obtained by optimally solving some restricted max-exposure instances in polynomial time using dynamic programming and carefully combining them to obtain an optimal solution in $(nm)^{O(h)}$ time for the case when input points lie within a horizontal strip of width h. Applying well known shifting techniques on this gives us the PTAS. Both bicriteria algorithms are obtained by carefully assigning the points to ranges and applying greedy strategies.

The remainder of this chapter is organized as follows. In Section 7.1, we discuss our hardness results followed by the bicriteria $O(k)$-approximation in Section 7.2. In Section 7.3, we study the case when $\mathcal{R}$ consists of translates of a fixed rectangle and describe a PTAS for it. Finally, in Section 7.4, we use these ideas to obtain a bicriteria $O(\sqrt{k})$-approximation when aspect ratio of rectangles in $\mathcal{R}$ is bounded by a constant.

7.1 Hardness of Max-Exposure

We show that the max-exposure problem for geometric ranges is both NP-hard, and inapproximable. We begin by reducing the densest k-subgraph on bipartite graphs (*bipartite-DkS*) to the max-exposure problem; the known NP-hardness of biparite-DkS then implies the hardness for max-exposure. Moreover, we show that bipartite-DkS is hard to approximate assuming the *dense vs random* conjecture, thereby establishing the inapproximability of max-exposure.

In the *bipartite-DkS* problem, we are given a bipartite graph $G = (A, B, E)$, an integer k, and we want to compute a set of k vertices such that the induced subgraph on those k vertices has the maximum number of edges. Given an instance $G = (A, B, E)$ of bipartite-DkS, we construct a max-exposure instance as follows.

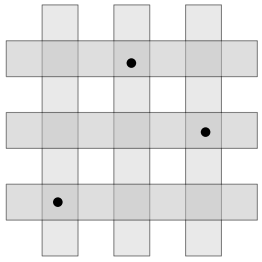


Figure 7.1: Reducing bipartite-DkS to max-exposure with axis-aligned rectangles.

Let $R_1 = [0, \epsilon] \times [0, n]$ be a thin vertical rectangle and $R_2 = [0, n] \times [0, \epsilon]$ be a thin horizontal rectangle. For each vertex $v_i \in A$, we create a copy R_i of R_1, and place it such that its lower-left corner is at $(i, 0)$. Similarly, for each vertex $v_j \in B$, we create a copy R_j of R_2, and place it such that its lower-left corner is at $(0, j)$. These $|A| + |B|$ rectangles create a checkerboard arrangement, with $|A| \times |B|$ cells of intersection. For each edge $(v_i, v_j) \in E$, we place a single point in the cell corresponding to intersection of R_i and R_j. It is now easy to see that G has a k-subgraph with m^* edges if and only if we can expose m^* points in this instance by removing k rectangles: the removed rectangles are

exactly the k vertices chosen in the graph, and each exposed point corresponds to the edge included in the output subgraph. (See also Figure 7.1.) We will later make use of this reduction, and therefore state it as the following lemma.

Lemma 63 *The max-exposure problem is at least as hard as bipartite-DkS.*

Since bipartite-DkS is known to be NP-hard [78], we have the following.

Theorem 26 *Max-exposure problem with axis-aligned rectangles is NP-hard.*

7.1.1 Hardness of Approximation

The construction in the preceding proof shows that max-exposure with rectangles is at least as hard as bipartite-DkS problem. Moreover, the geometric construction uses translates of *only two rectangles* R_1, R_2. In the following, we show that even with such a restricted range space, the problem is also hard to approximate. To that end we prove that bipartite-DkS cannot be approximated better than a factor $O(n^{1/4})$, where n is the number of vertices in this graph. More precisely, if the densest subgraph over k vertices has m^* edges, it is hard to find a subgraph over k vertices that contains $\Omega(m^*/n^{\frac{1}{4}-\epsilon})$ edges in polynomial time. This hardness of approximation is conditioned on the so-called *dense vs random* conjecture [10] stated as follows.

Given a graph G, constants $0 < \alpha, \beta < 1$, and a parameter k, we want to distinguish between the following two cases.

1. (Random) $G = G(n, p)$ where $p = n^{\alpha-1}$, that is, G has average degree approximately n^{α}.

2. (Dense) G is adversarially chosen so that the densest k-subgraph of G has average degree k^{β}.

The conjecture states that for all $0 < \alpha < 1$, sufficiently small $\epsilon > 0$, and for all $k \leq \sqrt{n}$, one cannot distinguish between the *dense* and *random* cases in polynomial time (w.h.p), when $\beta \leq \alpha - \epsilon$.

In order to obtain hardness guarantees using the above conjecture, one needs to find the 'distinguishing ratio' r, that is the least multiplicative gap between the optimum solution for the problem on the dense and random instances. If there exists an algorithm with an approximation factor significantly smaller than r, then we would be able to use it to distinguish between the dense and random instances, thereby refuting the conjecture. We obtain the following result for densest k-subgraph problem on bipartite graphs.

Lemma 64 *Assuming that* dense vs random *conjecture is true, the densest k-subgraph problem on bipartite graphs is hard to approximate better than a factor $O(n^{1/4})$ of optimum.*

Proof: Let $G' = (V', E')$ be a graph sampled either from the dense or the random instances. We construct a bipartite graph $G = (A, B, E)$ as follows. For every vertex $v \in V'$, add a vertex v_a to A and v_b to B. For every edge $e = (u, v) \in E'$, we add the pair of edges $e_1 = (u_a, v_b)$ and $e_2 = (v_a, u_b)$ to E. That is, every edge $e \in E'$ is mapped to two copies $e_1, e_2 \in E$ and we define e to be their *parent* edge as $par(e_1) = par(e_2) = e$. Similarly, for a vertex $u \in V'$ and its two copies $u_a, u_b \in V$, we define $par(u_a) = par(u_b) = u$. We say that G is *dense* if the underlying graph G' was sampled from the dense case, otherwise we say that G is *random*.

Consider a set of $k^* = 2k$ vertices in G. If G came from the dense case, there must be a set of $2k$ vertices that have $2k^{\beta+1}$ edges between them. So the number of edges in dense case $m_d^* \geq 2k^{\beta+1}$. Otherwise, we are in the random case. Consider the optimal set of $2k$ vertices V^* and let E^* be the set of edges in the induced subgraph $G[V^*]$. Now consider the corresponding set of vertices $V_p = \{par(v) \mid v \in V^*\}$ of the original graph G' and the set of edges E_p in the induced subgraph $G'[V_p]$). We have that

$|V_p| \leq |V^*| = 2k$ and $|E_p| \geq |E^*|/2$ because for each edge $e = (u, v) \in E^*$, we will have the edge $par(e) = (par(u), par(v)) \in E_p$. Since $|V_p| \leq 2k$ and we are in the random case, we can upperbound the number of edges in E_p as the number of edges in the densest subgraph of $G(n, n^{\alpha-1})$ over $2k$ vertices. This is $\tilde{O}(\max(2k, 4k^2 n^{\alpha-1}))$ w.h.p. where $\tilde{O}$ ignores logarithmic factors. Therefore the optimum number of edges in the random case is $m_r^* = |E^*| \leq 2|E_p| = \tilde{O}(\max(k, k^2 n^{\alpha-1}))$ w.h.p.

Choosing $k = n^{1/2}$, $\alpha = \frac{1}{2}$, $\beta = \frac{1}{2} - \epsilon$, gives us $m_r^* = \tilde{O}(n^{1/2})$ w.h.p. and $m_d^* = \tilde{\Omega}(n^{\frac{3-2\epsilon}{4}})$. Suppose, we could approximate this problem within a factor $O(n^{1/4-\epsilon})$, then in the dense case, the number of edges computed by this approximation algorithm is $\tilde{\Omega}(n^{\frac{1+\epsilon}{2}})$ which is strictly more than the maximum possible edges in the random case. Therefore, we would be able to distinguish between dense and random cases, and thereby refute the conjecture for these values of α, β and k. ■

Using the same construction as in Lemma 63, we obtain the following.

Corollary 27 *Assuming the dense vs random conjecture, max-exposure with axis-aligned rectangles is hard to approximate better than factor $O(n^{1/4})$ of optimum.*

7.1.2 Hardness of Max-exposure with Convex Polygons

We now show that the max-exposure problem is *equivalent* to the densest k-subhypergraph problem for *general hypergraphs* when the range space $(P, \mathcal{R})$ is defined by convex polygons. In one direction, the max-exposure instance $(P, \mathcal{R})$ naturally corresponds to a hypergraph $H = (\mathcal{R}, P)$ whose vertices are the ranges and the edges correspond to points and are defined by the containment relationship. Clearly, the densest k-subhypergraph corresponds to the set of k ranges deleting which exposes maximum number of points. For the other direction, we have the following lemma. (See also Figure 7.2.)

Lemma 65 *Given a hypergraph $H = (X, E)$, one can construct a max-exposure instance*

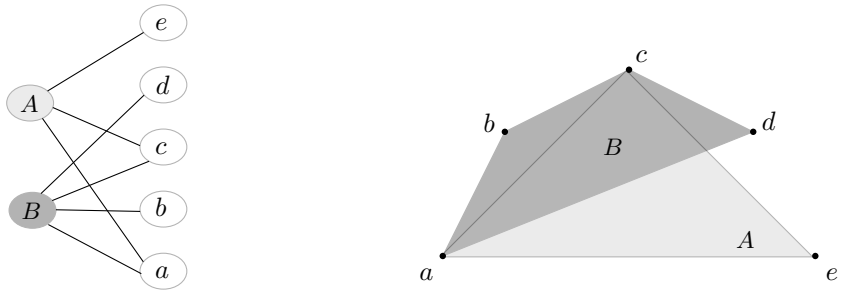


Figure 7.2: Reducing densest k-subhypergraph problem to max-exposure. Hypergraph vertices A, B shown as convex ranges.

with convex ranges $\mathcal{R}$ and points P such that the densest k-subhypergraph of H corresponds to a solution of max-exposure.

Proof: For each edge $e \in E$ of the hypergraph, add a point $p_e \in P$. We place all the points of P in convex position. Let $v \in X$ be a vertex and E_v be the set of hyperedges adjacent to v. Then for every $v \in X$, we add a convex polygon $R_v \in \mathcal{R}$ such that the corners of R_v is precisely the point set E_v. Note that this is possible since points of P are in convex position. It is easy to see that in order to include an edge e (expose p_e), we must include all vertices in E_v, which corresponds to removing all polygons corresponding to vertices in E_v. ■

7.2 A Bicriteria $O(k)$-approximation Algorithm

In this section, we present a simple approximation algorithm for the max-exposure problem that achieves bicriteria $O(k)$-approximation for range spaces defined by arbitrary axis-aligned rectangles. Specifically, if the optimal number of points exposed is m^*, the algorithm picks a subset of k^2 rectangles such that the number of points exposed is at least m^*/ck, for some constant c. In fact, the results hold for any polygonal range with $O(1)$ complexity.

This bicriteria approximation should be contrasted with the fact that no such approxi-

mation is possible for the densest k-subhypergraph problem: that is, one cannot compute a set of $O(k^b)$ vertices for any constant b such that the number of edges in the induced subhypergraph is at least optimal. Thus the geometric properties of the range space have a significant impact on the problem complexity. In particular, if $\mathcal{R}$ consists of rectangle ranges, we show that the following strategy picks a subset of αk ranges such that the number of points exposed is at least $\alpha m^*/(ck^2)$, for a parameter $1 \leq \alpha \leq k$ and constant c that will be fixed later. Choosing $\alpha = k$ gives us the claimed bound.

Our algorithm is essentially greedy. We divide the points into maximal equivalence classes, where each class is the maximal subset of points belonging to the same subset of ranges. We define $\mathcal{R}(p)$ as the set of ranges that contain a point $p \in P$, and remove all points that are contained in more than k ranges, since they can be never exposed in the optimal solution. Therefore, without loss of generality, we can assume that $|\mathcal{R}(p)| \leq k$ for all points $p \in P$. The rest of the algorithm is as follows.

Algorithm 6 Greedy-Bicriteria

1. Partition P into a set $\mathcal{G}$ of groups where each group $G_i \in \mathcal{G}$ is an equivalence class of points that are contained in the same set of ranges. That is, for any $p \in G_i,\ p' \in G_j$, we have $\mathcal{R}(p) = \mathcal{R}(p')$ if $i = j$ and $\mathcal{R}(p) \neq \mathcal{R}(p')$, otherwise.
2. Sort the groups in $\mathcal{G}$ by decreasing order of their size $|G_i|$ and select the ranges in first α groups for deletion.
3. Return $m' = \sum_{1 \leq i \leq \alpha} |G_i|$ as the number of points exposed.

In Algorithm 6, observe that every point in the ith group G_i is contained in the same set of ranges, denote it by $\mathcal{R}(G_i)$. Moreover, we have $|\mathcal{R}(G_i)| \leq k$. Therefore, the total number of ranges that we delete in Step 2 is at most αk. It remains to show that the number of points exposed m' is at least $\alpha m^*/ck^2$.

Lemma 66 *Let m' be the number of points exposed by the algorithm* Greedy-Bicriteria*,*

and let m^* be the optimal number of exposed points, Then, $m' \geq \alpha m^*/ck^2$.

Proof: Consider the optimal set $\mathcal{R}^*$ of k ranges that are deleted, and let P^* be the set of exposed points. We partition the set of points P^* into groups $\mathcal{G}^*$ as before, such that each group $G_i^* \in \mathcal{G}^*$ is identified by the range set $\mathcal{R}(G_i^*) = \mathcal{R}(p)$, for any $p \in G_i^*$. Since $P^* \subseteq P$, we must have that $\mathcal{G}^* \subseteq \mathcal{G}$. This holds because for every group $G_i^* \in \mathcal{G}^*$ there must be a group $G_j \in \mathcal{G}$ such that $\mathcal{R}(G_i^*) = \mathcal{R}(G_j)$. Moreover since P^* is the maximum set of points that can be exposed, we must have that $G_i^* = G_j$. Finally, we note that the number of groups $|\mathcal{G}^*|$ is bounded by the number of cells in the arrangement of ranges in $\mathcal{R}^*$ which is at most ck^2 for some fixed constant c, for all $O(1)$-complexity ranges. If the groups in $\mathcal{G}$ are arranged by decreasing order of their sizes, we have that

$$m^* = \sum_{1 \leq i \leq |\mathcal{G}^*|} |G_i^*| \leq \sum_{1 \leq i \leq |\mathcal{G}^*|} |G_i| \leq \sum_{1 \leq i \leq ck^2} |G_i|$$

$$\leq \frac{ck^2}{\alpha} \sum_{1 \leq i \leq \alpha} |G_i| = \frac{ck^2}{\alpha} \cdot m'$$

■

The parameter α can be tuned to improve the approximation guarantee with respect to one criterion (say the number of exposed points) at the cost of other. With $\alpha = k$, the algorithm exposes at least $\Omega(m^*/k)$ by removing k^2 ranges.

7.2.1 Constant Approximation for Bounded-ply Pseudodisks

If the range space $\mathcal{R}$ consists of pseudodisks of *bounded-ply* (no point in the plane is contained in more than a constant number ρ pseudodisks), then the algorithm Greedy-Bicriteria achieves a constant approximation. Due to the bounded-ply restriction, we

have that the number of pseudodisks containing the points of group G_i is $|\mathcal{R}(G_i)| \leq \rho$, and therefore number of pseudodisks that are removed in Step 2 of the algorithm is also at most $\alpha\rho$. Moreover, the number of cells in an arrangement of k pseudodisks with depth at most ρ is $O(\rho k)$ [22]. Therefore, we can bound the number of groups of the optimal solution $|\mathcal{G}^*|$ in the proof for Lemma 66 to be at most $c\rho k$. This gives us that the number of points exposed $m' \geq \alpha m^*/c\rho k$, where m^* is the number of points exposed by the optimal solution.

Lemma 67 *If the range space $\mathcal{R}$ consists of pseudodisks of* bounded-ply ρ*, then algorithm* Greedy-Bicriteria *exposes at least $\alpha m^*/c\rho k$ points by deleting at most $\alpha\rho$ pseudodisks, where $1 \leq \alpha \leq k$.*

Choosing $\alpha = k$, the algorithm achieves a bicriteria $O(\rho)$-approximation. With $\alpha = k/\rho$, the algorithm exposes at least $1/c\rho^2$ fraction of the optimal number of points by deleting k ranges.

7.3 A PTAS for Unit Square Ranges

We have seen that max-exposure is hard to approximate even if the ranges are translates of two types of rectangles. We now describe an approximation scheme when the ranges are translates of a *single* rectangle. In this case, we can scale the axes so that the rectangle becomes a *unit square* without changing any point-rectangle containment. Therefore, we can assume that our ranges are all unit squares. The problem is non-trivial even for unit square ranges, and as a warmup we first solve the following special case: *all the points lie inside a unit square.* We develop a dynamic programming algorithm to solve this case exactly, and then use it to design an approximation for the general set of points.

7.3.1 Exact Solution in a Unit Square

We are given a max-exposure instance consisting of unit square ranges $\mathcal{R}$ and a set of points P in a unit square C. Without loss of generality, we can assume that the lower left corner of C lies at origin $(0, 0)$ and all ranges in $\mathcal{R}$ intersect C. We classify the ranges in $\mathcal{R}$ to be one of the two types: (See also Figure 7.3).

Type-0 : Unit square ranges that intersect $x = 0$.

Type-1 : Unit square ranges that intersect $x = 1$.

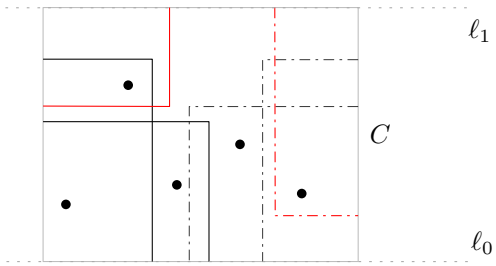


Figure 7.3: Max-exposure in a unit square C. Type 0 ranges are drawn with solid lines, Type 1 ranges are dash-dotted.

(A unit square range coincident with both $x = 0$ and $x = 1$ is assumed to be Type-0). We draw two parallel horizontal lines $\ell_0 : y = 0$ and $\ell_1 : y = 1$ coincident with bottom and top horizontal sides of C respectively. We say that a range $R \in \mathcal{R}$ is *anchored* to a line ℓ if it intersects ℓ. Note that every $R \in \mathcal{R}$ is anchored to exactly one of ℓ_0 or ℓ_1. (When R is coincident with both ℓ_0 and ℓ_1, we say that it is anchored to ℓ_0).

Moreover, for the rest of our discussion, let $x = x_i$ be a vertical line and define $P_i \subseteq P$ to be the set of points that have x-coordinate at least x_i. In other words, P_i is the set of *active points* at $x = x_i$. Similarly, define $\mathcal{R}_i \subseteq \mathcal{R}$ to be the set of *active ranges* that have at least one corner to the right of $x = x_i$. That is, $R \in \mathcal{R}_i$ either intersects $x = x_i$ or lies completely to the right of it.

In order to gain some intuition, we will first consider the following two natural dynamic programming formulations for the problem.

DP-template-0 Suppose that the points in P are ordered by their increasing x-coordinates and let x_i be the x-coordinate of the ith point p_i. We define a subproblem as $S(i, k', \mathcal{R}_d)$ which represents the maximum number of points in P_i that can be exposed by removing k' ranges from the range set $\mathcal{R}_i \setminus \mathcal{R}_d$. If we define $x_0 = 0$, then $S(0, k, \emptyset)$ gives the optimal number of exposed points for our problem.

Let $k_i = |\mathcal{R}(p_i) \setminus \mathcal{R}_d|$ be the number of new ranges in $\mathcal{R}_i$ that contain p_i. Then, we can can express the subproblems at i in terms of subproblems at $i + 1$ as follows.

$$S(i,\ k', \mathcal{R}_d) = \max \begin{cases} S(i+1,\ k' - k_i,\ \mathcal{R}_d \cup \mathcal{R}(p_i))\ +\ 1 & \text{expose } p_i \\ S\left(i+1,\ k',\ \mathcal{R}_d\right) & p_i \text{ not exposed} \end{cases}$$

Roughly speaking, at $x = x_i$ which is the *event* corresponding to a point $p_i \in P$, we have two choices : *expose* p_i or *do not expose* p_i. If we expose p_i, we pay for deleting the ranges in $\mathcal{R}_i \setminus \mathcal{R}_d$ that contain p_i and mark them as deleted by adding to the *deleted range set* $\mathcal{R}_d$. It is easy to see that this correctly computes the optimal number of exposed points since we charge for every deletion exactly once. However, there is one complication: a priori it is not clear how to bound the number of range subset $\mathcal{R}_d$ used by this dynamic program. We later argue that the geometry of range space for Type-0 ranges allows us to use only a polynomial number of choices.

DP-template-1 An alternative approach is to consider both *point* and *begin-range* events. That is, $x = x_i$ is either incident to a point $p_i \in P$ or to the left vertical side of a range $R_i \in \mathcal{R}$. Then, we can define a subproblem by the tuple $S(i, k', P_f)$ which represents the maximum number of points in $(P_i \setminus P_f)$ that can be exposed by removing

k' ranges in $\mathcal{R}_i$. If we define $x_0 = 0$, then $S(0, k, \emptyset)$ gives the optimal number of exposed points. Let $P(R_i) \subseteq P$ be the set of points contained in the range R_i, then we have the following recurrence.

$$S(i,\ k', P_f) \ = \ \max \begin{cases} S(i+1,\ k'-1,\ P_f) & \text{delete range } R_i \\ S(i+1,\ k',\ P_f \cup P(R_i)) & R_i \text{ not deleted} \end{cases}$$

(event $x = x_i$ was beginning of a range $R_i \in \mathcal{R}_i$)

$$= \ \max \begin{cases} S(i+1,\ k', P_f) & \text{if } p_i \in P_f, \text{ cannot expose } p_i \\ S(i+1,\ k', P_f) + 1 & \text{otherwise, expose } p_i \end{cases}$$

(otherwise, event $x = x_i$ was a point $p_i \in P_i$)

In the above formulation, at each *begin-range* event for some $R_i \in \mathcal{R}_i$, we have two choices: *delete* R_i or *do not delete* R_i. If R_i was deleted, we reduce the budget k' by one. Otherwise, if R_i was not deleted, we can never expose the points in $P(R_i)$, and therefore we add $P(R_i)$ to the *forbidden point set* P_f. The correctness of the dynamic program follows from the fact that for every point p_i, all the ranges containing it must begin before $x = x_i$, and we expose p_i only if those ranges were *deleted*. Again, it is not obvious how many different subsets P_f are needed by the dynamic program. However, we will later show that by keeping track of polynomial number of sets P_f, we can solve max-exposure with Type-1 ranges.

We note that the Type-0 and Type-1 ranges may superficially seem symmetric but once we fix the order of computing subproblems, they become structurally different. Therefore, we would need slightly different techniques to handle each type. For the ease of exposition, we present dynamic programs for Type-0 and Type-1 ranges separately and finally combine them. Also note that if the ranges in $\mathcal{R}$ are intervals on the real line (max

exposure in 1D), then both DP-template-0 and DP-template-1 can be easily applied to obtain a polynomial time algorithm.

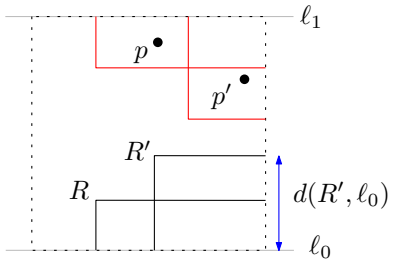


Figure 7.4: An example of *closer* relationship. Point p is *closer* to ℓ_1 than p'. R is closer to ℓ_0 than R'.

We will now define the following ordering relations that will be useful later. Let ℓ be a horizontal line, and let $d(p, \ell)$ denote the orthogonal distance of $p \in P$ from ℓ. If $p, p' \in P$ are two points, we say that p is *closer* to ℓ than p' if $d(p, \ell) < d(p', \ell)$. Similarly, for a range $R \in \mathcal{R}$ that is anchored to ℓ, let $d(R, \ell)$ be the vertical distance *inside the unit square* C between ℓ and the side of R parallel to ℓ. If $R, R' \in \mathcal{R}$ are two ranges, we say that R is *closer* (or equivalently R' is *farther*) from ℓ if both R, R' are anchored to ℓ and $d(R, \ell) < d(R', \ell)$. (See Figure 7.4.)

Max-exposure with Type-0 Ranges

Recall that Type-0 ranges intersect the vertical lines $x = 0$ and are anchored to either ℓ_0 or ℓ_1. We will apply the formulation discussed in *DP-template-0*. The key challenge here is to bound the number of possible deleted range sets $\mathcal{R}_d$. Towards that end, we make the following claim. Recall that $\mathcal{R}_i$ is the set of active ranges at $x = x_i$.

Lemma 68 *Let q_0, q_1 be the two exposed points strictly to the left of $x = x_i$ that are closest to ℓ_0 and ℓ_1 respectively. Then our dynamic program only needs to consider the set of deleted ranges $\mathcal{R}_d = \mathcal{R}_i \cap (\mathcal{R}(q_0) \cup \mathcal{R}(q_1))$ at $x = x_i$ conditioned on q_0, q_1.*

Proof: Observe that since $\mathcal{R}$ consists of Type-0 ranges, every range in $\mathcal{R}_i$ must intersect the vertical line $x = x_i$. Suppose we partition $\mathcal{R}_i$ into ranges $\mathcal{R}_i^0$ that are anchored to ℓ_0 and $\mathcal{R}_i^1$ that are anchored to ℓ_1. Let $P' \subseteq P$ be the set of all *exposed points* strictly to the left of $x = x_i$. Observe that for all $p \in P'$, any range $R \in \mathcal{R}_i^0$ that contains p must also contain q_0. Therefore, we must have $\mathcal{R}_i^0 \cap \mathcal{R}(p) \subseteq \mathcal{R}_i^0 \cap \mathcal{R}(q_0)$, for all $p \in P'$. Similarly, $\mathcal{R}_i^1 \cap \mathcal{R}(p) \subseteq \mathcal{R}_i^1 \cap \mathcal{R}(q_1)$, for all $p \in P'$. This gives us $\bigcup_{p \in P'} \mathcal{R}_i \cap \mathcal{R}(p) = \mathcal{R}_i \cap (\mathcal{R}(q_0) \cup \mathcal{R}(q_1))$. Therefore, the set $\mathcal{R}_d$ consists of all the active ranges that contain at least one exposed point and were therefore deleted to the left of $x = x_i$. ■

Therefore, if our dynamic program remembers the exposed points q_0, q_1, then we can compute the deleted range set $\mathcal{R}_d = \mathcal{R}_i \cap (\mathcal{R}(q_0) \cup \mathcal{R}(q_1))$ at $x = x_i$. There are $O(m^2)$ choices for the pair q_0, q_1, so the number of possible sets $\mathcal{R}_d$ is also $O(m^2)$. We can therefore identify our subproblems by the tuple $S(i,\ k',\ q_0,\ q_1)$ which represents the maximum number of exposed points with x-coordinates x_i or higher using k' rectangles from the set $\mathcal{R}_i \setminus \mathcal{R}_d$. With $k_i = |\mathcal{R}(p_i) \setminus \mathcal{R}_d|$, we obtain the following recurrence:

$$S(i,\ k',\ q_0,\ q_1) \ = \ \max \begin{cases} S\left(i+1,\ k'-k_i,\ \mathit{closer}(p_i, q_0),\ \mathit{closer}(p_i, q_1)\right) \ + \ 1 & \text{expose } p_i \\ S\left(i+1,\ k',\ q_0,\ q_1\right) & p_i \text{ not exposed} \end{cases}$$

where the function $\mathit{closer}(p_i, q_0)$ returns whichever of p_i, q_0 is closer to ℓ_0, and $\mathit{closer}(p_i, q_1)$ returns whichever of p_i, q_1 is closer to ℓ_1. The optimal solution is given by $S(0,\ k,\ q_0^*,\ q_1^*)$, where $q_0^* = (0, 1)$ and $q_1^* = (0, 0)$ are two artificial points with $\mathcal{R}(q_0^*) = \mathcal{R}(q_1^*) = \emptyset$ (not contained in any range). The base case is defined by the rightmost event at vertical line $x = 1$ and is initialized with zeroes for all q_0, q_1 and $k' \geq 0$. Any subproblem with $k' < 0$ has value $-\infty$.

Max-exposure with Type-1 Ranges

Next we consider the case when we only have Type-1 ranges in $\mathcal{R}$. Unfortunately in this case, our previous dynamic program does not work and we need to remember a different set of parameters. More precisely, we will apply the formulation discussed in *DP-template-1*, and bound the number of possible forbidden point sets P_f. Recall that P_i is the set of active points at $x = x_i$ (with x-coordinate x_i or higher).

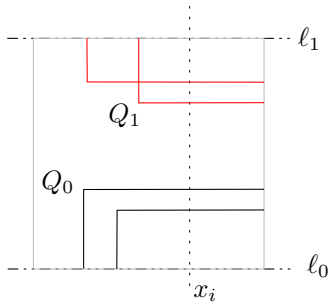


Figure 7.5: Undeleted ranges Q_0 and Q_1 farthest from ℓ_0 and ℓ_1 respectively.

Lemma 69 *Let Q_0, Q_1 be two ranges that begin to the left of $x = x_i$ and were* not deleted. *Moreover, Q_0 is anchored to and is farthest from ℓ_0. Similarly Q_1 is anchored to and is farthest from ℓ_1 (Figure 7.5). Then the forbidden point set at $x = x_i$ is given by $P_f = P_i \cap (P(Q_0) \cup P(Q_1))$, where $P(Q)$ is the set of points contained in range Q.*

Proof: Recall that the set $\mathcal{R}_i$ consists of ranges that have at least one corner to the right of the vertical line $x = x_i$. Since we are dealing with Type-1 ranges, every range that begins to the left of $x = x_i$ lies in $\mathcal{R}_i$. Now let $\mathcal{R}' \subseteq \mathcal{R}_i$ be the set of ranges that begin to the left of $x = x_i$ and were *not deleted*. Recall that P_i is the set of points in P that have x-coordinate x_i or higher. Now consider any range $R \in \mathcal{R}'$. Observe that if R was anchored to ℓ_0, then every point of P_i that lies in R also lies in Q_0. Otherwise, if R was anchored to ℓ_1, every point of P_i that lies in R also lies in Q_1. Therefore,

$\bigcup_{R\in\mathcal{R}'} (P_i \cap P(R)) = P_i \cap (P(Q_0) \cup P(Q_1))$, which is precisely the forbidden point set P_f. ■

Therefore, if our dynamic program remembers the ranges Q_0 and Q_1, we can compute the forbidden point set $P_f = P_i \cap (P(Q_0) \cup P(Q_1))$ at $x = x_i$. Since there are $O(n^2)$ choices for the pair Q_0, Q_1, the number of possible sets P_f is also $O(n^2)$. We can now identify the subproblems by the tuple $S(i,\ k',\ Q_0,\ Q_1)$ which represents the maximum number of points in $P_i \setminus P_f$ that are exposed by deleting k' ranges that begin on or after $x = x_i$. This gives us the following recurrence.

$$S(i,\ k',\ Q_0,\ Q_1)\ =$$

$$\max \begin{cases} S(i+1,\ k'-1,\ Q_0,\ Q_1) & \text{delete range } R_i \\ S(i+1,\ k',\ \mathit{farther}(R_i, Q_0),\ \mathit{farther}(R_i, Q_1)) & R_i \text{ not deleted} \end{cases}$$

$$(\textit{event } x = x_i \textit{ was beginning of a range } R_i \in \mathcal{R})$$

$$\max \begin{cases} S(i+1,\ k',\ Q_0,\ Q_1) & \text{if } p_i \in P_f, \text{ cannot expose } p_i \\ S(i+1,\ k',\ Q_0,\ Q_1) + 1 & \text{otherwise, expose } p_i \end{cases}$$

$$\textit{(otherwise, event } x = x_i \textit{ was a point } p_i \in P\textit{)}$$

Here, the function *farther* simply updates the ranges Q_0, Q_1 with R_i if needed. More precisely, if R_i is anchored to ℓ_0 and is farther from ℓ_0 than Q_0, then $\mathit{farther}(R_i, Q_0)$ returns R_i, otherwise it returns Q_0. Similarly, if R_i is anchored to ℓ_1, and is farther from ℓ_1 than Q_1, then $\mathit{farther}(R_i, Q_1)$ returns R_i, otherwise it returns Q_1.

The optimal solution is given by $P(0, k, Q_0^*, Q_1^*)$, where Q_0^*, Q_1^* are two artificial ranges of zero-width : Q_0^* is anchored to ℓ_0 and is defined by corners $(0,0)$ and $(0,1)$; similarly, Q_1^* is anchored to ℓ_1 and is defined by corners $(0,1)$ and $(1,1)$.

Figure 7.6: Remembering one of R_1, R_2 in (a) or one of p_1, p_2 in (b) is not sufficient.

Remark 1 We note that remembering a constant number of exposed points q_0, q_1 (DP-template-0) or a constant number of undeleted ranges Q_1, Q_2 (DP-template-1) by themselves cannot solve *both* Type-0 and Type-1 ranges. For instance, in Figure 7.6(a) with Type-0 ranges, if R_1, R_2 were both *not deleted* but we remembered one of them, then we will incorrectly expose one of p, p'. Similarly in Figure 7.6(b) with Type-1 ranges, if p_1, p_2 were both exposed but we only remembered one of them, we will pay for one of the ranges R, R' again when we expose p_3. However, since both the dynamic programs for Type-0 and Type-1 ranges express subproblems at event i in terms of subproblems at event $i+1$, we can easily combine them with minor adjustments.

Combining them together

In the following, we will combine the dynamic programs for Type-0 and Type-1 ranges to obtain a dynamic program for max-exposure in a unit square C. We will need a couple of changes. First, the events at $x = x_i$ are now defined by either a point $p_i \in P$ or beginning of a Type-1 range R_i. Next, the *deleted range set* $\mathcal{R}_d$ at $x = x_i$ will only consist of Type-0 ranges and is defined as $\mathcal{R}_d = \mathcal{R}_{i0} \cap (\mathcal{R}(q_0) \cup \mathcal{R}(q_1))$ where $\mathcal{R}_{i0} \subseteq \mathcal{R}_i$ is the set of Type-0 ranges that intersect the vertical line $x = x_i$. The *forbidden point set* $P_f = P_i \cap (P(Q_0) \cup P(Q_1))$ stays the same. Here q_0, q_1, Q_0, Q_1 are same as defined before. The subproblems represent the maximum number of points in $P_i \setminus P_f$ that can be exposed by deleting k' ranges from $\mathcal{R}_i \setminus \mathcal{R}_d$. If $k_i = |(\mathcal{R}(p_i) \cap \mathcal{R}_{i0}) \setminus \mathcal{R}_d|$, then we obtain

the following combined recurrence.

$$S(i,\ k',\ q_0,\ q_1, Q_0, Q_1)\ =$$

$$\max \begin{cases} S(i+1,\ k',\ q_0,\ q_1,\ Q_0,\ Q_1) & \text{if } p_i \in P_f, \text{ cannot expose } p_i \\ S(i+1,\ k',\ q_0,\ q_1,\ Q_0,\ Q_1) & \text{choose to not expose } p_i \\ S(i+1,\ k'-k_i,\ \mathit{closer}(q_0,p_i),\ \mathit{closer}(q_1,p_i),\ Q_0,\ Q_1)+1 & \text{expose } p_i \end{cases}$$

$$(\textit{event } x = x_i \textit{ was a point } p_i \in P_i)$$

$$\max \begin{cases} S(i+1,\ k'-1,\ q_0,\ q_1,\ Q_0,\ Q_1) & \text{delete Type-1 range } R_i \\ S(i+1,\ k',\ q_0,\ q_1,\ \mathit{farther}(R_i,Q_0),\ \mathit{farther}(R_i,Q_1)) & R_i \text{ not deleted} \end{cases}$$

$$(\textit{event } x = x_i \textit{ was beginning of a Type-1 range } R_i \in \mathcal{R}_i)$$

The optimal solution is given by $S(0,\ k,\ q_0^*,\ q_1^*,\ Q_0^*,\ Q_1^*)$. The correctness of the above formulation follows from the fact that when we choose to expose p_i, we are guaranteed that all Type-1 ranges in $\mathcal{R}(p_i)$ have already been deleted, and the expression k_i only charges for Type-0 ranges containing p_i. As for the running time, for each event $x = x_i$, we compute $O(kn^2m^2)$ entries and computing each entry takes constant time. Since there are $O(n+m)$ events, we obtain the following.

Lemma 70 *Given a set P of m points in a unit square C and a set of n unit square ranges $\mathcal{R}$, we can compute their max-exposure in $O(k(n+m)n^2m^2)$ time.*

7.3.2 A Constant Factor Approximation

We now use the preceding algorithm to solve the max-exposure problem for general set of points and unit square ranges within a factor 4 of optimum. In particular, we compute a set of $4k$ ranges in $\mathcal{R}$ such that the number of points exposed in P by deleting them is

at least the optimal number of points. Suppose we embed the ranges $\mathcal{R}$ on a uniform unit-sized grid G, and define $\mathcal{C}$ as the collection of all cells in G that contain at least one point of P. Then we can solve exactly for each cell in $\mathcal{C}$ and combine them using dynamic programming as described in Algorithm 7 (DP-Approx). See also Figure 7.7.

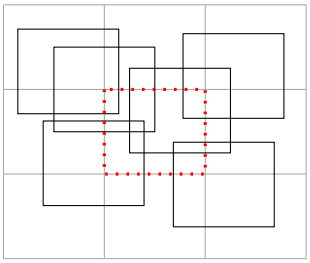


Figure 7.7: Embedding a max-exposure instance with unit square ranges on a unit-sized grid. Optimal solution in each grid cell can be computed exactly using Lemma 70.

Algorithm 7 DP-Approx

1. Apply Lemma 70 to solve max-exposure locally in every cell $C_i \in \mathcal{C}$ for all $0 \leq k_i \leq k$. Call this a *local* solution denoted by $local(P(C_i), \mathcal{R}(C_i), k_i)$, where $P(C_i) \subseteq P$ is the set of points contained in cell C_i and $\mathcal{R}(C_i)$ is the set of ranges intersecting C_i.

2. Process cells in $\mathcal{C}$ in any order $C_1, C_2, \ldots, C_g$, and define $global(i,\ k')$ as the maximum number of points exposed in the cells C_i through C_g using k' ranges. Combine local solutions to obtain $global(i,\ k')$ as follows.

$$global(i,\ k') \ = \ \max_{0 \leq k_i \leq k'} \ global(i+1,\ k'-k_i) \ + \ local(P(C_i),\ \mathcal{R}(C_i),\ k_i)$$

3. Return $global(1, 4k)$ as the number of exposed points.

Lemma 71 *If $P^* \subseteq P$ is the optimal set of exposed points, then $global(1, 4k) \geq |P^*|$, that is , the algorithm* DP-Approx *achieves a 4-approximation and runs in $O(k(n+m)n^2m^2)$ time.*

Proof: Consider the optimal set of ranges $\mathcal{R}^* \subseteq \mathcal{R}$. Observe that each range $R \in \mathcal{R}^*$ intersects at most four grid cells. Let $R_i = R \cap C_i$ be the rectangular region defined by

intersection of R and C_i. Clearly, there are at most four regions R_i for each $R \in \mathcal{R}^*$ and therefore $4k$ in total. At this point, the regions in cell C_i are disjoint from regions in some other cell $C_j \in \mathcal{C}$. Therefore, optimal solution exposes $|P^*|$ points over a set of cells $\mathcal{C}^*$ such that the set $\mathcal{R}^*$ has at most $4k$ disjoint components in the cells $\mathcal{C}^*$. Since we can solve the problem exactly for each cell and can combine them using the above dynamic program, we have that $\mathit{global}(1, 4k) \geq |P^*|$ and we achieve a 4-approximation.

For the running time, we observe that solving max-exposure locally in a cell C_i takes $O(k(n_i + m_i)n_i^2 m_i^2)$ time, where n_i is the number of ranges that intersect C_i and m_i is the number of points in P that lie in C_i. Summed over all cells, we get the following bound.

$$\begin{aligned}\sum_i k(n_i + m_i)n_i^2 m_i^2 \;&\leq\; k\;\sum_i (n_i + m_i)\;\sum_i n_i^2\;\sum_i m_i^2 \\ &\leq k(n+m)\;(\sum_i n_i)^2\;(\sum_i m_i)^2 \;=\; O(k(n+m)n^2m^2)\end{aligned}$$

Once the local solutions are computed, the dynamic program that merges them into a global solution has $O(k|\mathcal{C}|)$ subproblems and computing each subproblem takes $O(k)$ time. Recall that every cell in $\mathcal{C}$ contains at least one point, so $|\mathcal{C}| \leq n$ and the merge step takes an additional $O(k^2 n)$ time. ■

7.3.3 Towards a PTAS

We now consider the max-exposure instance in a horizontal strip of unit width. That is, all points in P lie in a horizontal strip bounded by lines ℓ_0, ℓ_1 and $\mathcal{R}$ consists of unit square ranges. Suppose, we subdivide the strip into unit square cells $C_1, C_2, \ldots, C_r \in \mathcal{C}$ ordered from left to right. We make the following simple observation.

Lemma 72 *Let $R \in \mathcal{R}$ be a unit square range and C_{j-1} be the first cell from left which it intersects. Then the only other cell that R can intersect is C_j. Moreover, R is Type-1 with*

respect to C_{j-1} and Type-0 with respect to C_j. (See Figure 7.8.)

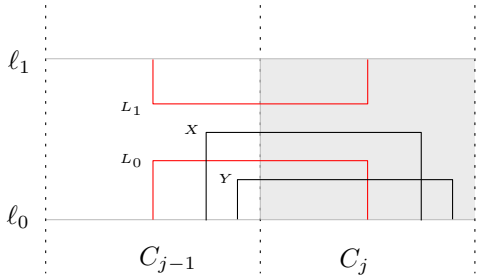


Figure 7.8: Max-exposure instance in a strip. $C_{j-1}, C_j \in \mathcal{C}$ are two consecutive cells. All ranges in the figure are Type-1 in cell C_{j-1} and Type-0 in cell C_j. For L_0, L_1 highlighted in the figure, we have $\mathcal{L}_{>0} = \{X\}$ and $\mathcal{L}_{>1} = \emptyset$. Moreover, both ranges L_0 and X *dominate* the range Y.

Observe that the set of points exposed in cell C_j will also depend on the set of Type-0 objects of C_j that were already deleted in C_{j-1}. So we need to ensure that we do not double count the set of ranges that were already deleted in C_{j-1}. To do this, we again use a dynamic program similar to that for max-exposure within a cell where we express the subproblems at $x = x_i$ in terms of subproblems to the right of $x = x_i$. However, there are some important differences in how we define our subproblems. First, events at a vertical line $x = x_i$ are one of three types:

1. *cell-boundary*: $x = x_i$ is coincident with left-boundary of a cell $C_j \in \mathcal{C}$,
2. *begin-range*: $x = x_i$ is coincident with left-vertical side of a range $R_i \in \mathcal{R}$
3. *point*: $x = x_i$ is incident to a point $p_i \in P$

Moreover for a given cell C_j, in addition to the points q_0, q_1, and ranges Q_0, Q_1, we will also need to remember two additional ranges : L_0 (anchored to ℓ_0) and L_1 (anchored to ℓ_1) that begin in C_{j-1}, were *not deleted* and are farthest from ℓ_0, ℓ_1 respectively. For the sake of clarity, we will use $Z_0 = (q_0, Q_0, L_0)$ to denote the triplets corresponding to ℓ_0 and $Z_1 = (q_1, Q_1, L_1)$ to denote the triplets corresponding to ℓ_1.

Suppose $x = x_i$ lies in the cell C_j. Then we show that the set of deleted ranges $\mathcal{R}_d$ consisting of Type-0 ranges in C_j, and the set of forbidden points P_f can be uniquely identified using the triples Z_0, Z_1.

- *Deleted Type-0 range-set* $\mathcal{R}_d$ Let $\mathcal{R}_{j-1}$ be the set of ranges that begin in cell C_{j-1}, and therefore are Type-1 with respect to C_{j-1}. Suppose we define $\mathcal{L}_{>0} \subseteq \mathcal{R}_{j-1}$ to be the set consisting of ranges anchored to ℓ_0 and farther from ℓ_0 than L_0. Similarly, $\mathcal{L}_{>1} \subseteq \mathcal{R}_{j-1}$ consists of ranges anchored to ℓ_1 and farther from ℓ_1 than L_1. Then, we define $\mathcal{R}_d = \mathcal{R}_{i0} \cap (\mathcal{R}(q_0) \cup \mathcal{R}(q_1) \cup \mathcal{L}_{>0} \cup \mathcal{L}_{>1})$. Recall that $\mathcal{R}_{i0} \subseteq \mathcal{R}_i$ is the set of active Type-0 ranges. (See also Figure 7.8.)

- *Forbidden point-set* $P_f = P_i \cap (P(L_0) \cup P(L_1) \cup P(Q_0) \cup P(Q_1))$.

Finally, we say that a range R *dominates* another range R', if both R, R' begin in the same cell C_j and $R' \cap C_j \subseteq R \cap C_j$. That is, R completely contains the part of R' that lies in cell C_j (Figure 7.8). Note that the key difference from earlier formulations is that at a begin-range event for a Type-1 range R_i in cell C_j, we choose to *ignore* R_i if it is dominated by ranges Q_0 or Q_1, because the points of R_i contained in C_j already lie in the forbidden set P_f. With $k_i = |(\mathcal{R}(p_i) \cap \mathcal{R}_{i0}) \setminus \mathcal{R}_d|$, we obtain the following recurrence.

$$S(i,\ k',\ Z_0,\ Z_1) \ =\ S(i+1,\ k,\ \mathcal{U}(Z_0, C_j),\ \mathcal{U}(Z_1, C_j)) \qquad \text{update } L_0, L_1$$

(*event* $x = x_i$ *is left-boundary of cell* C_j)

$$\max \begin{cases} S(i+1,\ k',\ Z_0,\ Z_1) & \text{if } p_i \in P_f, \text{ cannot expose } p_i \\ S(i+1,\ k',\ Z_0,\ Z_1) & \text{choose to not expose } p_i \\ S(i+1,\ k' - k_i,\ \mathcal{U}(Z_0, p_i),\ \mathcal{U}(Z_1, p_i)\ ) + 1 & \text{expose } p_i, \text{ update } q_0, q_1 \end{cases}$$

(*otherwise, event* $x = x_i$ *was a point* p_i *in cell* C_j)

$$\max \begin{cases} S(i+1,\ k',\ Z_0,\ Z_1) & \text{if either } Q_0 \text{ or } Q_1 \text{ dominates } R_i, \textit{ ignore } R_i \\ S(i+1,\ k'-1,\ Z_0,\ Z_1) & \text{delete Type-1 range } R_i \\ S(i+1,\ k',\ \mathcal{U}(Z_0, R_i),\ \mathcal{U}(Z_1, R_i)) & R_i \text{ not deleted, update } Q_0 \text{ or } Q_1 \end{cases}$$

(*otherwise, event* $x = x_i$ *was beginning of a Type-1 range* R_i *in cell* C_j)

The function $\mathcal{U}(Z,\ E)$ used above is defined as follows. Roughly speaking, it updates the triplets $Z \in \{Z_0, Z_1\}$ based on the event E and returns an updated triplet. We have the following three cases.

- For a cell-boundary event C_j, if we have $Z_0 = (q_0, Q_0, L_0)$, the function $\mathcal{U}(Z_0, C_j) = (q_0^*,\ Q_0^*,\ Q_0)$. Similarly, $\mathcal{U}(Z_1, C_j) = (q_1^*,\ Q_1^*,\ Q_1)$. This corresponds to resetting the points q_0, q_1, rectangles Q_0, Q_1 for the current cell C_j, and remembering the rectangles Q_0, Q_1 from the previous cell C_{j-1} as L_0, L_1 respectively.

- For a point event p_i, we have $\mathcal{U}(Z_0, p_i) = (\textit{closer}(p_i, q_0),\ Q_0,\ L_0)$ and similarly $\mathcal{U}(Z_1, p_i) = (\textit{closer}(p_i, q_1),\ Q_1,\ L_1)$. Recall that the function $\textit{closer}(p_i, q_0)$ returns whichever of p_i, q_0 is closer to ℓ_0, and $\textit{closer}(p_i, q_1)$ returns whichever of p_i, q_1 is closer to ℓ_1.

- Finally for a begin-rectangle event R_i, we have the following two updates: $\mathcal{U}(Z_0, R_i) = (q_0,\ \textit{farther}(R_i, Q_0),\ L_0)$ for triplet Z_0 and $\mathcal{U}(Z_1, R_i) = (q_1,\ \textit{farther}(R_i, Q_1),\ L_1)$ for Z_1. Recall that the function $\textit{farther}(R_i, Q_0)$ returns R_i, if R_i is anchored to and is farther from ℓ_0 than Q_0. Similarly, $\textit{farther}(R_i, Q_1)$ returns R_i if R_i is anchored to and is farther from ℓ_1 than Q_1.

The optimal solution is given by $W(0,\ k,\ Z_0^\emptyset,\ Z_1^\emptyset)$ where $Z_0^\emptyset = (q_0^*,\ Q_0^*,\ Q_0^*)$ and $Z_1^\emptyset = (q_1^*,\ Q_1^*,\ Q_1^*)$. In order to establish the correctness of the above formulation, we make the following claim.

Lemma 73 *Let $P^* \subseteq P$ be the optimal set of exposed points. Then, for every point $p_i \in P^*$, we count the range $R \in \mathcal{R}(p_i)$ towards the total number of deleted ranges exactly once.*

Proof: We begin by noting that R intersects at most two cells : C_{j-1} as a Type-1 range and C_j as a Type-0 range. It suffices to show that we count R towards the total number of deleted ranges in exactly one of these two cells. Alternatively, it suffices to show that we count R in cell C_j if and only if we have not already counted R in C_{j-1}. Recall that we can only count for R in C_{j-1} by deleting it at a begin-range event. Moreover, we can only count for R in C_j when a point $p_i \notin P_f$ that lies in cell C_j is exposed. Without loss of generality, assume that R is anchored to ℓ_0. The case when R is anchored to ℓ_1 is symmetric.

We first consider the easy case when R was not deleted in C_{j-1}. Observe that since R is Type-0 with respect to C_j, similar to the earlier cases, the terms $\mathcal{R}(q_0) \cup \mathcal{R}(q_1)$ in the expression for $\mathcal{R}_d$ will correctly charge for R in cell C_j.

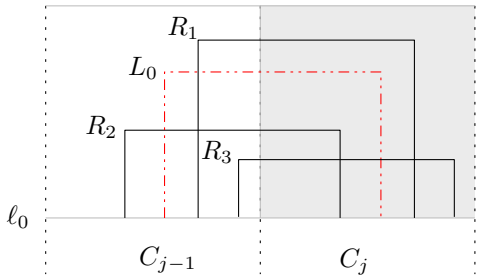


Figure 7.9: Three cases for the proof: $R_1 \in \mathcal{L}_{>0}$, and $R_2, R_3 \notin \mathcal{L}_{>0}$. R_2 begins before L_0 and R_3 begins after L_0.

Now, we move to the second case where we are currently in cell C_j and we have already counted R by deleting it at a begin-range event in cell C_{j-1}. In this case, we show that we will not count R again in C_j. More precisely, we show that if R contains an exposed point p that lies in cell C_j but *is not contained* in the forbidden point set P_f, then the deleted

range set $\mathcal{R}_d$ contains R, and therefore the expression for $k_i = |(\mathcal{R}(p_i) \cap \mathcal{R}_{i0}) \setminus \mathcal{R}_d|$ will not charge for R again. We have three cases.

1. $R \in \mathcal{L}_{>0}$. This case is straightforward as $\mathcal{R}_d$ contains all ranges in $\mathcal{L}_{>0}$.

2. $R \notin \mathcal{L}_{>0}$ *and* R *begins before* L_0. This case is not possible because any point that is contained in $(R \cap C_j)$ is also contained in L_0. This holds because R and L_0 have the same width, so if R begins before L_0 in C_{j-1}, it must end before L_0 in C_j. Since every point contained in L_0 is contained in the forbidden set P_f, we must have $p \in P_f$ which is a contradiction. (See Figure 7.9 with $R = R_2$.)

3. $R \notin \mathcal{L}_{>0}$ *and* R *begins after* L_0. This case is also not possible because if this were true L_0 would have dominated R. Therefore, we would have *ignored* R in C_{j-1} and would not have deleted it. (See Figure 7.9 with $R = R_3$.)

■

Lemma 74 *The restricted max-exposure instance such that all points in P lie within a unit-width horizontal strip bounded by lines ℓ_0, ℓ_1 and $\mathcal{R}$ consists of unit squares can be solved in $O(k(n+m)n^4m^2)$ time, where $m = |P|$ and $n = |\mathcal{R}|$.*

Using similar ideas as Lemma 71, this readily gives a 2-approximation for max-exposure. More precisely, we can embed the input instance on to a unit-sized grid as before, but instead of solving max-exposure in a cell, we use the above algorithm to solve max-exposure locally in a row of the grid. Since each range $R \in \mathcal{R}$ can intersect at most two rows, R is split into two sub-ranges R_1, R_2 contained in at most two rows. Since these new sub-ranges in two different rows are disjoint, there exists an optimal solution with $2k$ sub-ranges. Therefore, if we have already computed the local solutions for each row i, using the algorithm *DP-Approx* we can compute *global*$(1, 2k)$ which exposes at least optimal number of points using at most $2k$ ranges.

Corollary 28 *There exists a 2-approximation algorithm for max-exposure with unit square ranges running in $O(k(n+m)n^4m^2)$ time.*

Generalizing to h anchor lines The dynamic program for max-exposure in a horizontal strip bounded by two anchor lines ℓ_0, ℓ_1 can be generalized to the case when we have h anchor lines $\ell_1, \ell_2, \ldots, \ell_h$. However, there is a minor technical change required. Observe that for a given anchor line ℓ_i, there can be points and anchored ranges on either side of ℓ_i. Therefore, we will need to remember the closest exposed points and the farthest undeleted ranges on both sides of ℓ_i. So for each anchor line ℓ_i, we will need the triplet $Z_i^+ = (q_i^+, Q_i^+, L_i^+)$ for points and ranges above ℓ_i and the triplet $Z_i^- = (q_i^-, Q_i^-, L_i^-)$ for points and ranges below ℓ_i. The dynamic program will now need to remember at most $4h$ ranges and $2h$ points which gives a running time of $O(k(n+m)n^{4h}m^{2h})$. If we denote a collection of h consecutive anchor lines by a *bundle* of width h, then we have the following lemma that will be used later.

Lemma 75 *Max-exposure in a bundle of width h can be solved in $O(k(n+m)n^{4h}m^{2h})$ time.*

7.3.4 An $(1+\epsilon)$-Approximation Algorithm

We are now ready to describe our PTAS for the problem. Suppose the anchor lines correspond to the horizontal lines of the uniform unit-sized grid G. Since we have already solved max-exposure exactly for h consecutive rows in G, we can now apply standard shifting techniques [79] to obtain an $(1+\epsilon)$-approximation. If P^* is the optimal set of exposed points, then we show how to compute a set of $(1+\epsilon)k$ ranges deleting which will expose at least $|P^*|$ points. Note that using similar ideas but with a little more work, it is also possible to expose at least $(1-\epsilon)|P^*|$ points by deleting exactly k ranges. We show this later in Section 7.3.5.

Suppose that anchor lines $\ell_1, \ell_2, \ldots, \ell_z$ are ordered by increasing y-coordinates. We define a *bundle* B_j to be a set of h consecutive anchor lines, identified by the lowest index anchor ℓ_j. We also define *bundle-set* to be a sequence of consecutive bundles, identified by the index of the lowest bundle. For instance the bundle B_1 comprises of anchor lines ℓ_1 through ℓ_h (inclusive). And the bundle-set $\mathcal{B}_1$ comprises of bundles $B_1, B_h, B_{2h}, \ldots B_{\lceil z/h \rceil}$. The lines $\ell_1, \ell_h, \ldots, \ell_{\lceil z/h \rceil}$ form the *bundle boundaries* $\partial\mathcal{B}_1$ of bundle-set $\mathcal{B}_1$.

For each bundle $B_j \in \mathcal{B}_1$, we can use the dynamic program from Lemma 75 to solve max-exposure locally. Using the exact solution for each bundle as local solution, we can use the algorithm DP-Approx (from Section 7.3.2) to combine them into a global solution for the bundle-set $\mathcal{B}_1$ given by $P(\mathcal{B}_1) = \mathit{global}(1, (k + k/h))$. We repeat this for each bundle-set $\mathcal{B}_i$ for all $i \in \{1, 2, \ldots, h\}$, and return the point set $P(\mathcal{B}_i)$ that has maximum cardinality over all $i \in \{1, 2, \ldots, h\}$.

It remains to show that this achieves a good approximation. To see this, we observe that the only ranges that may be double counted are the ones that are anchored to bundle boundaries of $\partial\mathcal{B}_i$. In the following, we show that this number is a small fraction of the optimum solution.

Lemma 76 *The bundle boundaries $\partial\mathcal{B}_i, \partial\mathcal{B}_j$ for any two bundle-set $\mathcal{B}_i, \mathcal{B}_j$ are disjoint, and therefore the set of ranges anchored to lines in $\partial\mathcal{B}_i$ are also disjoint. Then, there exists a bundle-set $\mathcal{B}_{\min}$ such that the number of ranges of the optimal solution anchored to lines in $\partial\mathcal{B}_{\min}$ is at most k/h.*

Proof: Let $\mathcal{R}^* \subseteq \mathcal{R}$ be the optimal set of ranges, and let $\mathcal{R}^*(\partial\mathcal{B}_i) \subseteq \mathcal{R}^*$ be the set of ranges anchored to lines in $\partial\mathcal{B}_i$. Since $\bigcup_{i\in\{1,\ldots h\}} \partial\mathcal{B}_i$ is the set of all anchor lines, we have

$$\bigcup_{i\in\{1,\ldots h\}} \mathcal{R}^*(\partial\mathcal{B}_i) \;=\; \mathcal{R}^* \qquad \Longrightarrow \sum_{i\in\{1,\ldots h\}} |\mathcal{R}^*(\partial\mathcal{B}_i)| \;=\; k$$

$$\implies \sum_{i\in\{1,\ldots h\}} |\mathcal{R}^*(\partial\mathcal{B}_{\min})| \;\leq\; k \qquad \implies |\mathcal{R}^*(\partial\mathcal{B}_{\min})| \;\leq\; k/h$$

■

Choosing $\epsilon = 1/h$ gives us a set of $(1+\epsilon)k$ objects such that the number of points exposed by selecting these objects is at least the optimum number of points.

Theorem 29 *There exists an $(1+\epsilon)$-approximation algorithm for max-exposure with unit square ranges running in $O(k(n+m)n^{4/\epsilon}m^{2/\epsilon})$ time.*

7.3.5 An Alternative $(1+\epsilon)$-Approximation Algorithm

Given a set of points P, unit square ranges $\mathcal{R}$, we will now show that the PTAS for unit square ranges can be modified so that we can compute a set of k ranges that expose at least $(1-\epsilon)$ fraction of the maximum possible number of points. For simplicity we assume that h is odd. The basic setup is the same: we have the anchor lines $\ell_1, \ell_2, \ldots, \ell_z$ that are unit distance apart. However, there is one important change, we will only use the odd-numbered lines $\ell_1, \ell_3, \ldots, \ell_h, \ell_{h+2}, \ldots, \ell_z$ to define bundles. For instance, the bundle B_1 now consists of the anchor lines $\ell_1, \ell_3, \ldots, \ell_h$, while the bundle-set $\mathcal{B}_1$ now comprises of bundles $B_1, B_h, B_{2h}, \ldots, B_{z/h}$. Same as before, the lines $\ell_1, \ell_h, \ldots, \ell_{z/h}$ form the boundary $\partial\mathcal{B}_1$. We have the following algorithm.

Clearly, the number of ranges used by the above algorithm is k. It remains to show that the number of points m' exposed by the algorithm is also close to m^*, the optimal number of exposed points. Let $P^* \subseteq P$ be the optimal set of exposed points.

Lemma 77 *The bundle boundaries $\partial\mathcal{B}_i, \partial\mathcal{B}_j$ for any two bundle-set $\mathcal{B}_i, \mathcal{B}_j$ are disjoint, and therefore the set of points assigned to lines in $\partial\mathcal{B}_i$ are also disjoint. Then, there exists a bundle-set $\mathcal{B}_{\min}$ such that the number of points of P^* assigned to its boundaries $\partial\mathcal{B}_{\min}$ is at most $\frac{2m^*}{h-1}$.*

Algorithm 8 PTAS-Exposed-Points

1. Assign each point $p \in P$ to the closest line among $l_1, l_3, \ldots l_z$.
2. For each $i \in \{1, 3, \ldots, h\}$, process bundle set $\mathcal{B}_i$ as follows.
   - Let A_i be the set of points assigned to anchor lines $l_j \in \partial\mathcal{B}_i$, boundaries of $\mathcal{B}_i$.
   - Using the exact algorithm for each bundle $B \in \mathcal{B}_i$ as local solutions, we run the algorithm DP-Approx (from Section 7.3.2) over the point set $P \setminus A_i$ to obtain global solutions given by $global(1, k)$. Let $P(\mathcal{B}_i)$ be the set of exposed points returned by DP-Approx.
3. Return the set $P(\mathcal{B}_i)$ that has maximum cardinality over all $i \in \{1, 3, \ldots, h\}$.

Proof: let $P^*(\partial\mathcal{B}_i) \subseteq P^*$ be the set of points in P^* that are assigned to lines in boundaries $\partial\mathcal{B}_i$ of some bundle $\mathcal{B}_i$. Since $\bigcup_{i\in\{1,3,\ldots,h\}} \partial\mathcal{B}_i$ is the set of all anchor lines to which we assign points, we have

$$\bigcup_{i\in\{1,3,\ldots h\}} P^*(\partial\mathcal{B}_i) = P^* \implies \sum_{i\in\{1,3\ldots h\}} |P^*(\partial\mathcal{B}_i)| = m^*$$

$$\implies \sum_{i\in\{1,3,\ldots h\}} |P^*(\partial\mathcal{B}_{\min})| \leq m^* \implies \left(\frac{h-1}{2}\right) |P^*(\partial\mathcal{B}_{\min})| \leq m^*$$

$$\implies |P^*(\partial\mathcal{B}_{\min})| \leq \frac{2m^*}{h-1}$$

■

Observe that for the bundle-set $\mathcal{B}_{\min}$, we may have removed $A_{\min} = P^*(\partial\mathcal{B}_{\min})$ points, but the remaining set $P \setminus A_{\min}$ consists at least $m^* - \frac{2m^*}{h-1} = (1 - \frac{2}{h-1})m^*$ points of the optimal set P^*. Moreover, observe that we have removed points that are within a unit distance on either side of anchor line $\ell_j \in \partial\mathcal{B}_{\min}$, the set of ranges deleted in each bundle are disjoint from another. Therefore, the value $P(\mathcal{B}_{\min})$ returned by the algorithm DP-Approx exposes at least $P \setminus A_{\min} = (1 - \frac{2}{h-1})m^*$ points by deleting k ranges. If we set $h = 2/\epsilon + 1$ we have the following result.

Theorem 30 *There exists an $(1-\epsilon)$-approximation on the number of exposed points for max-exposure with unit-square ranges running in $k(nm)^{O(1/\epsilon)}$ time.*

7.4 Extensions and Applications

In this section, we discuss some extensions and applications of our the results from previous section. We say that the range family $\mathcal{R}$ consists of *fat rectangles* if every range $R \in \mathcal{R}$ is a rectangle of bounded *aspect ratio*. Moreover, we say that $\mathcal{R}$ consists of *similar and fat rectangles*, if ranges in $\mathcal{R}$ are rectangles and the ratio of the largest to the smallest side in $\mathcal{R}$ is constant. We show that if $\mathcal{R}$ consists of *similar and fat* rectangles, one can achieve a constant approximation. Moreover, if $\mathcal{R}$ consists of *fat rectangles* one can achieve a bicriteria $O(\sqrt{k})$-approximation.

7.4.1 Approximation for Similar and Fat Rectangles

Let a, b be the length of smallest and largest sides of rectangles in $\mathcal{R}$ such that $b/a = c$ is constant. Then we can modify the input instance as follows. Replace each range $R \in \mathcal{R}$ by tiling it with at most c^2 squares of sidelength a such that the area occupied by R and its replacements are the same. Now, we have a modified set of ranges $\mathcal{R}'$ consisting of squares that have the same sidelength. Consider the optimal solution with k ranges $\mathcal{R}^*$ that exposes m^* points. It is easy to see that the set $\mathcal{R}^*$ corresponds to at most c^2k ranges in the modified instance, and therefore deleting c^2k ranges from $\mathcal{R}'$ exposes at least m^* points. Therefore, we can run the polynomial-time 2-approximation algorithm (Corollary 28) to obtain a set of at most $2c^2k$ ranges that expose at least m^* points.

Theorem 31 *Given a set of points P, a set of rectangle ranges $\mathcal{R}$ such that the ratio of largest to smallest side in $\mathcal{R}$ is bounded by a constant, then there exists a polynomial time*

$O(1)$-*approximation algorithm for max-exposure.*

7.4.2 Approximation for Fat Rectangles

We now consider the case when rectangles in $\mathcal{R}$ have bounded aspect ratio. That is for all rectangles $R \in \mathcal{R}$, the ratio of its two sides is bounded by a constant c. We transform the input ranges $\mathcal{R}$ to obtain a modified set of ranges $\mathcal{R}'$ as follows. For each rectangle $R \in \mathcal{R}$, let x be the length of the smaller side of R. Then we replace R by at most $\lceil c \rceil$ *squares* each of sidelength x. If m^* is the optimal number of points exposed by deleting k ranges from $\mathcal{R}$, then there exists a set of $O(k)$ ranges in $\mathcal{R}'$ deleting which will expose at least m^* points. Observe that the set $\mathcal{R}'$ consists of *square* ranges, of possibly different sizes. Therefore, if we can obtain an f-approximation for square ranges, we can easily obtain $O(f)$-approximation with fat rectangles.

A Bicriteria $O(\sqrt{k})$-approximation for Squares

We will describe an approximation algorithm for the case when the set of ranges $\mathcal{R}$ consists of axis-aligned squares. We achieve an approximation algorithm in three steps. First, we partition the point set by assigning them to one of the input squares. Next, we solve the problem exactly for a fixed square. Finally, we combine these solutions to achieve a good approximation to the optimal solution.

We define $\mathcal{A} : P \to \mathcal{R}$ to be a function that assigns a point in P to exactly one range in $\mathcal{R}$. If $\mathcal{R}(p_i)$ is the set of squares that contain p_i, then $\mathcal{A}(p_i)$ is the smallest square in $\mathcal{R}(p_i)$. This assignment scheme ensures the following property.

Lemma 78 *Let $R \in \mathcal{R}$ be a square and let $\mathcal{P}_R = \mathcal{A}^{-1}(R)$ be the set of points* assigned *to it. Moreover, let $\mathcal{R}' \subseteq \mathcal{R}$ be the set of squares that intersect R and contain at least one point in $\mathcal{P}_R$. Then, every square $R' \in \mathcal{R}'$ must have sidelength bigger than that of R, and*

therefore contains at least one corner of R.

Now suppose we fix a square R, and consider a restricted max-exposure instance with the set of its assigned points $\mathcal{P}_R$. Since, ranges that contain a point in $\mathcal{P}_R$ are all bigger then R, this case is essentially the same as points inside a unit square, and therefore Lemma 70 can be easily extended to solve it exactly. This gives us the following algorithm. Here $1 \leq \alpha \leq k$ is a parameter.

Algorithm 9 Greedy-Squares

1. For every square $R \in \mathcal{R}$, apply Lemma 70 over the point set $\mathcal{P}_R$ to expose the maximum set of points $P(R,k) \subseteq \mathcal{P}_R$ by deleting k ranges.
2. Order squares in $\mathcal{R}$ by decreasing $|P(R,k)|$ values, and pick the set $\mathcal{S} \subseteq \mathcal{R}$ of first α squares.
3. Return $\bigcup_{R \in \mathcal{S}} P(R,k)$ as the set of exposed points.

Lemma 79 *Let m^* be the optimal number of points exposed using k squares, then algorithm* Greedy-Squares *computes a set of at most αk squares that expose at least $\alpha m^*/k$ points.*

Proof: It is easy to see that the number of squares is at most αk. To show the bound on number of points exposed, consider the optimal set $\mathcal{R}^*$ of k ranges and let the optimal set of points exposed by $\mathcal{R}^*$ to be P^*. We will now use the same assignment procedure $\mathcal{A}^* : P^* \to \mathcal{R}^*$ to assign points in P^* to a square in $\mathcal{R}^*$. That is, $\mathcal{A}^*(p_i)$ is the smallest square in $\mathcal{R}^*$ that contains p_i. We claim that $\mathcal{A}^*(p_i) = \mathcal{A}(p_i)$ for all $p_i \in P^*$ since every square that contains p_i lies in $\mathcal{R}^*$. Moreover, let $\mathcal{P}^*_R$ denote the set of points of P^* assigned to R.

Let m' be the number of points exposed by the algorithm and assume that the squares in $\mathcal{R}$ are ordered such that $|P(R_i,k)| \geq |P(R_j,k)|$ for all $i < j$. Then, we have the

following.

$$\begin{aligned} m^* &= \left| \bigcup_{R \in \mathcal{R}^*} \mathcal{P}_R^* \right| = \sum_{R \in \mathcal{R}^*} |\mathcal{P}_R^*| \\ &\leq \sum_{1 \leq i \leq k} |P(R_i, k)| \leq \frac{k}{\alpha} \sum_{1 \leq i \leq \alpha} |P(R_i, k)| = \frac{k}{\alpha} m' \end{aligned}$$

■

For $\alpha = \sqrt{k}$, the above algorithm achieves a bicriteria $O(\sqrt{k})$-approximation. Since an f-approximation for square ranges gives an $O(f)$-approximation for fat rectangles, we obtain the following.

Theorem 32 *Given a set of points P and a set of ranges $\mathcal{R}$ consisting of rectangles of bounded aspect ratio, then one can obtain a bicriteria $O(\sqrt{k})$-approximation for max-exposure in polynomial time.*

7.5 Bibliographic Notes

Coverage and exposure problems have been widely studied in geometry and graphs. In the classical Set Cover problem, we want to select a subfamily of k sets that cover the maximum number of items (points) [80, 81]. For the set cover problem, the classical greedy algorithm achieves a factor $\log n$ approximation for the number of sets needed to cover all the items, or factor $(1 - 1/e)$ approximation for the number of items covered by using exactly k sets. Similarly, in geometry, the art gallery problems explore coverage of polygons using a minimum number of guards. Unlike coverage problems where greedy algorithms deliver reasonably good approximation, the exposure problems turn out to be much harder. Specifically, choosing k sets whose union is of *minimum size* is much harder to approximate with a conditional inapproximability of $O(n^{1-\epsilon})$ where n is the number of elements, or $O(m^{1/4-\epsilon})$ where m is the number of sets [10]. This so-called

min-union problem is essentially the densest k-subgraph problem on hypergraphs [82]. The densest k-subgraph problem for graphs has a long history [83, 84, 85, 11]. The paper [82] also studies the special case of an *interval hypergraph* $H = (V, E)$, whose vertices V is a finite subset of $\mathbb{N}$ and for each edge $e \in E$ there are values $a_e, b_e \in \mathbb{N}$ such that $e = \{i \in V : a_e \leq i \leq b_e\}$. That is, vertices are integer points and edges are intervals containing them. They show that this restricted case can be solved in polynomial time. The corresponding max exposure instance is when ranges $\mathcal{R}$ are intervals $R_i = (a_i, b_i)$ on the real line. As discussed in the chapter, this 1-D case can also be solved in polynomial time.

The coverage problems have also been studied for geometric set systems where improved approximation bounds are possible using the VC dimension [86, 87, 88]. Multi-cover variants, where each input point must be covered by more than one set, are studied in [89, 90]. The minimum constraint removal problem [7, 91] (studied in Chapter 3), where given a set of ranges, the goal is to *expose* a path between two given points by deleting at most k ranges (a path is exposed if it lies in the exterior of all ranges), is also closely related to the max-exposure problem. Even for simple shapes such as unit disks (or unit squares) [16, 92], no PTAS is known for this problem.

Chapter 8

Conclusion and Open Problems

Motivated by applications to robotics, sensor networks and path planning under uncertainty, the goal of this dissertation was to study some path finding and exposure problems in the plane under the notion of constraint violation. We made progress towards this goal by making clean theoretical formulations for these problems, and designing a family of non-trivial approximation and exact algorithms for them. Although we leave some questions unanswered, this dissertation will hopefully serve as an important first step in the study of these class of problems.

We began our discussion with the study of min-color path problem in Chapter 2, where we studied lowerbounds (hardness guarantees) and upperbounds (approximation algorithms) for the problem of finding a minimum color path in a colored graph. Assuming plausible complexity conjectures, we prove a lowerbound of $O(n^{1/4})$ and an upperbound of $O(n^{1/2})$ on vertex-colored graphs. Similarly, we obtain a lowerbound of $O(n^{1/3})$ and an upperbound of $O(n^{2/3})$ for edge-colored graphs. A natural open problem to consider is the following.

Open Problem 1 (Min-Color Path) *Is it possible to design tight approximation algorithms for minimum color path on vertex- and edge-colored graphs?*

The log-density framework has been useful in designing tight approximation bounds for related problems such as minimum k-union [10] and densest k-subgraph [11]. It would be interesting to see if those techniques can be applied to min-color path.

In Chapters 3 and 4, we studied the problem of designing approximation algorithms for minimum constraint removal (MCR). We first designed an $O(\sqrt{n})$ approximation algorithm for MCR and later improved it to obtain an $O(\log n)$ approximation in Chapter 4. As for the lowerbounds, we showed that MCR is as hard to approximate as vertex cover, which cannot be approximated to a factor significantly better than 2 assuming unique games conjecture. A natural open question is the following.

Open Problem 2 (MCR) *Is it possible to design tight approximation algorithms for the minimum constraint removal problem?*

Although, we have reasons to believe that MCR exhibits a constant approximation algorithm but finding the best possible constant remains a challenging open problem.

In Chapters 5 and 6, we study the problem of shortest path among removable and pairwise disjoint polygonal obstacles in the plane. We showed that if k is the number of obstacles to be removed, the problem can be solved in $O(k^2 n \log n)$ time under euclidean distances and $O(kn \log^2 n)$ time under manhattan distances. These algorithms were then subsequently applied to obtain fast fully polynomial approximation schemes (FPTAS) for the problem of obstacles with weighted costs and a target cost budget. A natural question to consider here is to get rid of the multiplicative dependence on k and design near-linear algorithms for $k = \Theta(n)$. Also recall that the polynomial time solvability crucially depends on the obstacles being convex. So a natural question to understand the complexity of the problem with non-convex but pairwise disjoint obstacles. This gives us the following two problems.

Open Problem 3 (Shortest Obstacle Removing Paths)

1. *Can we design a near-linear time algorithm for the case of disjoint convex obstacles in the plane?*

2. *What happens if we allow the obstacles to be non-convex? Does the problem becomes NP-hard? Can we design an algorithm that computes a path which removes $O(k)$ obstacles and computes a path no longer than the shortest path removing k obstacles?*

We believe that the problem is likely NP-hard with non-convex obstacles. Designing a good approximation algorithm remains a challenging open problem.

Finally, in Chapter 7 we studied the maximum exposure problem that is motivated by reliability of coverage in geometric networks. We show that the problem is hard to approximate even for simple shapes such as axis-aligned rectangles. For unit squares, we give a PTAS and use it to obtain constant approximation algorithms for some restricted cases. However the following questions remain open.

Open Problem 4 (Maximum Exposure) *Is the max-exposure problem with squares of arbitrary side length (or arbitrary disks) NP-hard? Does there exists a constant approximation for this case?*

Recall that if the *overlap number* of pseudodisks (disks or squares) is bounded, we gave a constant approximation in Section 7.2.1. Can we say the same if the overlap is not bounded by a constant?

Bibliography

[1] L. Erickson and S. LaValle, *A simple, but np-hard, motion planning problem*, in *Proceedings of AAAI, AAAI press*, 2013.

[2] K. R. Tseng and D. G. Kirkpatrick, *On barrier resilience of sensor networks*, in *7th ALGOSENSORS 2011*, pp. 130–144, 2011.

[3] H. Alt, S. Cabello, P. Giannopoulos, and C. Knauer, *On some connection problems in straight-line segment arrangements*, *27th EuroCG* (2011) 27–30.

[4] E. Eiben and I. Kanj, *How to navigate through obstacles?*, *CoRR* **abs/1712.04043** (2017).

[5] M. Korman, M. Löffler, R. I. Silveira, and D. Strash, *On the complexity of barrier resilience for fat regions*, in *9th ALGOSENSORS 2013*, pp. 201–216, 2013.

[6] N. Kumar, *Computing a minimum color path in edge-colored graphs*, in *Analysis of Experimental Algorithms*, (Cham), pp. 35–50, Springer International Publishing, 2019.

[7] S. Bandyapadhyay, N. Kumar, S. Suri, and K. Varadarajan, *Improved approximation bounds for the minimum constraint removal problem*, in *Proceedings of 21st APPROX*, pp. 2:1–2:19, 2018.

[8] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri, *Computing Shortest Paths in the Plane with Removable Obstacles*, in *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, vol. 101, pp. 5:1–5:15, 2018.

[9] N. Kumar, S. Sintos, and S. Suri, *The maximum exposure problem*, in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[10] E. Chlamtáč, M. Dinitz, and Y. Makarychev, *Minimizing the union: Tight approximations for small set bipartite vertex expansion*, in *Proceedings of the 28th SODA*, pp. 881–899, 2017.

[11] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, *Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph*, in *Proceedings of the 42nd STOC*, pp. 201–210, ACM, 2010.

[12] E. Chlamtac, M. Dinitz, and R. Krauthgamer, *Everywhere-sparse spanners via dense subgraphs*, in *Proceedings of the 53rd FOCS*, pp. 758–767, 2012.

[13] E. Chlamtáč, P. Manurangsi, D. Moshkovitz, and A. Vijayaraghavan, *Approximation algorithms for label cover and the log-density threshold*, in *Proceedings of the 28th SODA*, pp. 900–919, 2017.

[14] G. Goel, C. Karande, P. Tripathi, and L. Wang, *Approximability of combinatorial problems with multi-agent submodular cost functions*, in *Proceedings of the 50th FOCS*, pp. 755–764, 2009.

[15] S. Yuan, S. Varma, and J. P. Jue, *Minimum-color path problems for reliability in mesh networks*, in *24th INFOCOM 2005*, vol. 4, pp. 2658–2669, IEEE, 2005.

[16] S. Bereg and D. G. Kirkpatrick, *Approximating barrier resilience in wireless sensor networks*, in *5th ALGOSENSORS 2009*, pp. 29–40, 2009.

[17] S. Jha, O. Sheyner, and J. Wing, *Two formal analyses of attack graphs*, in *Computer Security Foundations Workshop*, pp. 49–63, IEEE, 2002.

[18] R. Hassin, J. Monnot, and D. Segev, *Approximation algorithms and hardness results for labeled connectivity problems*, *J. Comb. Optim.* **14** (2007), no. 4 437–453.

[19] M. R. Fellows, J. Guo, and I. Kanj, *The parameterized complexity of some minimum label problems*, *Journal of Computer and System Sciences* **76** (2010), no. 8 727–740.

[20] S. O. Krumke and H.-C. Wirth, *On the minimum label spanning tree problem*, *Information Processing Letters* **66** (1998), no. 2 81–85.

[21] H. J. Broersma, X. Li, G. Woeginger, and S. Zhang, *Paths and cycles in colored graphs*, *Australasian journal of combinatorics* **31** (2005), no. 1 299–311.

[22] K. Clarkson and P. Shor, *Application of random sampling in computational geometry, II*, *Discrete & Computational Geometry* **4** (1989) 387–421.

[23] K. Kedem, R. Livne, J. Pach, and M. Sharir, *On the union of jordan regions and collision-free translational motion amidst polygonal obstacles*, *Discrete & Computational Geometry* **1** (1986) 59–70.

[24] S.Khot and O. Regev, *Vertex cover might be hard to approximate to within 2-epsilon*, *Journal of Computer and System Sciences* **74** (2008), no. 3 335–349.

[25] T. M. Chan and E. Grant, *Exact algorithms and apx-hardness results for geometric packing and covering problems*, *Computational Geometry* **47** (2014), no. 2 112–124.

[26] D. Y. C. Chan and D. G. Kirkpatrick, *Multi-path algorithms for minimum-colour path problems with applications to approximating barrier resilience*, *Theor. Comput. Sci.* **553** (2014) 74–90.

[27] K. Hauser, *The minimum constraint removal problem with three robotics applications*, in *Tenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2012*, pp. 1–17, 2012.

[28] I. K. E. Eiben, J. Gemmell and A. Youngdahl, *Improved results for minimum constraint removal*, in *Proceedings of AAAI, AAAI press*, 2018.

[29] T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, *Journal of the ACM (JACM)* **46** (1999), no. 6 787–832.

[30] J. Hershberger and S. Suri, *An optimal algorithm for Euclidean shortest paths in the plane*, *SIAM Journal on Computing* **28** (1999), no. 6 2215–2256.

[31] D. Kirkpatrick, *Optimal search in planar subdivisions*, *SIAM Journal on Computing* **12** (1983), no. 1 28–35.

[32] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, *Optimal point location in a monotone subdivision*, *SIAM Journal on Computing* **15** (1986), no. 2 317–340.

[33] H. Rohnert, *Shortest paths in the plane with convex polygonal obstacles*, *Information Processing Letters* **23** (1986), no. 2 71–76.

[34] S. Eriksson-Bique, J. Hershberger, V. Polishchuk, B. Speckmann, S. Suri, T. Talvitie, K. Verbeek, and H. Yıldız, *Geometric k shortest paths*, in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1616–1625, 2015.

[35] T. Asano, *An efficient algorithm for finding the visibility polygon for a polygonal region with holes*, *IEICE TRANSACTIONS (1976-1990)* **68** (1985), no. 9 557–559.

[36] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, *Visibility of disjoint polygons*, *Algorithmica* **1** (1986), no. 1-4 49–63.

[37] S. K. Ghosh and D. M. Mount, *An output-sensitive algorithm for computing visibility graphs*, *SIAM Journal on Computing* **20** (1991), no. 5 888–910.

[38] S. Kapoor and S. N. Maheshwari, *Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles*, in *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pp. 172–182, 1988.

[39] M. H. Overmars and E. Welzl, *New methods for computing visibility graphs*, in *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pp. 164–171, 1988.

[40] J. S. B. Mitchell, *A new algorithm for shortest paths among obstacles in the plane*, *Annals of Mathematics and Artificial Intelligence* **3** (1991), no. 1 83–105.

[41] J. S. B. Mitchell, *Shortest paths among obstacles in the plane*, *International Journal of Computational Geometry & Applications* **6** (1996), no. 3 309–332.

[42] J. A. Storer and J. H. Reif, *Shortest paths in the plane with polygonal obstacles*, *Journal of the ACM (JACM)* **41** (1994), no. 5 982–1012.

[43] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, *Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons*, *Algorithmica* **2** (1987), no. 1-4 209–233.

[44] D. T. Lee and F. P. Preparata, *Euclidean shortest paths in the presence of rectilinear barriers*, *Networks* **14** (1984), no. 3 393–410.

[45] J. Hershberger and J. Snoeyink, *Computing minimum length paths of a given homotopy class*, *Computational Geometry* **4** (1994), no. 2 63–97.

[46] J. S. B. Mitchell and C. H. Papadimitriou, *The weighted region problem: finding shortest paths through a weighted planar subdivision*, *Journal of the ACM (JACM)* **38** (1991), no. 1 18–73.

[47] J. Hershberger, S. Suri, and H. Yıldız, *A near-optimal algorithm for shortest paths among curved obstacles in the plane*, in *Proceedings of the Twenty-Ninth Annual Symposium on Computational Geometry*, pp. 359–368, 2013.

[48] D. Z. Chen and H. Wang, *Computing shortest paths among curved obstacles in the plane*, *ACM Trans. Algorithms* **11** (2015), no. 4 26:1–26:46.

[49] T. M. Chan, *Low-dimensional linear programming with violations*, *SIAM Journal on Computing* **34** (2005), no. 4 879–893.

[50] T. Roos and P. Widmayer, *k-violation linear programming*, *Information Processing Letters* **52** (1994), no. 2 109–114.

[51] J. Matoušek, *On geometric optimization with few violated constraints*, *Discrete & Computational Geometry* **14** (1995), no. 4 365–384.

[52] S. Har-Peled and V. Koltun, *Separability with outliers*, *16th International Symposium on Algorithms and Computation* (2005) 28–39.

[53] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.

[54] M. Farshi, P. Giannopoulos, and J. Gudmundsson, *Improving the stretch factor of a geometric network by edge augmentation*, *SIAM Journal on Computing* **38** (2008), no. 1 226–240.

[55] J.-L. D. Carufel, C. Grimm, A. Maheshwari, and M. Smid, *Minimizing the continuous diameter when augmenting paths and cycles with shortcuts*, in *15th Scandinavian Symposium and Workshops on Algorithm Theory*, pp. 27:1–27:14, 2016.

[56] M. Abellanas, A. Garca, F. Hurtado, J. Tejel, and J. Urrutia, *Augmenting the connectivity of geometric graphs*, *Computational Geometry* **40** (2008), no. 3 220 – 230.

[57] A. Maheshwari, S. C. Nandy, D. Pattanayak, S. Roy, and M. Smid, *Geometric path problems with violations*, *Algorithmica* (2016) 1–24.

[58] J. Hershberger, N. Kumar, and S. Suri, *Shortest paths in the plane with obstacle violations*, in *Proc. 25th Annual Eur. Symp. on Alg.*, vol. 87, pp. 49:1–49:14, 2017.

[59] D. Z. Chen and H. Wang, *A new algorithm for computing visibility graphs of polygonal obstacles in the plane*, *J. Comput. Geom.* **6** (2015), no. 1 316–345.

[60] K. Clarkson, S. Kapoor, and P. Vaidya, *Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time*, in *Proc. 3rd Annual Symp. Comput. Geom.*, pp. 251–257, ACM, 1987.

[61] N. Sarnak and R. E. Tarjan, *Planar point location using persistent search trees*, *Communic. ACM* **29** (1986), no. 7 669–679.

[62] T. M. Chan and Y. Nekrich, *Towards an optimal method for dynamic planar point location*, in *Proc. 56th Symp. Found. Comp. Science*, pp. 390–409, IEEE, 2015.

[63] D. Lee, C.-D. Yang, and T. Chen, *Shortest rectilinear paths among weighted obstacle*, *Int. J. Comput. Geom. & Appl.* **1** (1991), no. 02 109–124.

[64] D. Lee, C. Yang, and C. Wong, *Rectilinear paths among rectilinear obstacles*, *Discrete Applied Mathematics* **70** (1996), no. 3 185–215.

[65] T. Feder, R. Motwani, L. O'Callaghan, C. Olston, and R. Panigrahy, *Computing shortest paths with uncertainty*, *J. Algorithms* **62** (2007), no. 1 1–18.

[66] Y. Gao, *Shortest path problem with uncertain arc lengths*, *Computers & Mathematics with Applications* **62** (2011), no. 6 2591–2600.

[67] P. Kamousi, T. M. Chan, and S. Suri, *Stochastic minimum spanning trees in Euclidean spaces*, in *Proc. 27th Annual Symp. Comput. Geom.*, pp. 65–74, ACM, 2011.

[68] E. Nikolova, M. Brand, and D. R. Karger, *Optimal route planning under uncertainty*, in *Proc. 16th Int. Conf. Autom. Plann. and Sched.*, vol. 6, pp. 131–141, 2006.

[69] E. M. Arkin, J. S. Mitchell, and C. D. Piatko, *Bicriteria shortest path problems in the plane*, in *Proc. 3rd Canad. Conf. Comput. Geom*, pp. 153–156, 1991.

[70] D. Z. Chen, O. Daescu, and K. S. Klenk, *On geometric path query problems*, *Int. J. Comp. Geom. & Applic.* **11** (2001), no. 06 617–645.

[71] H. Wang, *Bicriteria rectilinear shortest paths among rectilinear obstacles in the plane*, in *Proc. 33rd Annual Symp. Comput. Geom.*, pp. 60:1–60:16, 2017.

[72] C. Yang, D. Lee, and C. Wong, *On bends and lengths of rectilinear paths: a graph-theoretic approach*, *Int. J. Comput. Geom. & Appl.* **2** (1992), no. 01 61–74.

[73] C. Yang, D. Lee, and C. Wong, *Rectilinear path problems among rectilinear obstacles revisited*, *SIAM J. Comput.* **24** (1995), no. 3 457–472.

[74] D. Z. Chen, K. S. Klenk, and H. T. Tu, *Shortest path queries among weighted obstacles in the rectilinear plane*, *SIAM J. Comput.* **29** (2000), no. 4 1223–1246.

[75] M. Iwai, H. Suzuki, and T. Nishizeki, *Shortest path algorithm in the plane with rectilinear polygonal obstacles*, in *Proc. SIGAL Workshop*, 1994.

[76] Y. Chiang and J. Mitchell, *Two-point Euclidean shortest path queries in the plane*, in *Proc. 10th ACM-SIAM Annual Symp. Discrete Algorithms*, SIAM, 1999.

[77] D. Z. Chen, R. Inkulu, and H. Wang, *Two-point L_1 shortest path queries in the plane*, in *Proc. 30th Annual Symp. Comput. Geom.*, p. 406, ACM, 2014.

[78] U. Feige and M. Seltser, *On the densest k-subgraph problems*, tech. rep., Weizmann Institute of Science, Jerusalem, Israel, 1997.

[79] D. S. Hochbaum and W. Maass, *Approximation schemes for covering and packing problems in image processing and vlsi*, *Journal of the ACM (JACM)* **32** (1985), no. 1 130–136.

[80] U. Feige, *A threshold of ln n for approximating set cover*, *Journal of the ACM (JACM)* **45** (1998), no. 4 634–652.

[81] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, *Optimal packing and covering in the plane are np-complete*, *Information processing letters* **12** (1981), no. 3 133–137.

[82] E. Chlamtác, M. Dinitz, C. Konrad, G. Kortsarz, and G. Rabanca, *The densest k-subhypergraph problem*, *SIAM Journal on Discrete Mathematics* **32** (2018), no. 2 1458–1477.

[83] U. Feige, D. Peleg, and G. Kortsarz, *The dense k-subgraph problem*, *Algorithmica* **29** (2001), no. 3 410–421.

[84] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama, *Greedily finding a dense subgraph*, *Journal of Algorithms* **34** (2000), no. 2 203–221.

[85] S. Arora, D. Karger, and M. Karpinski, *Polynomial time approximation schemes for dense instances of np-hard problems*, *Journal of computer and system sciences* **58** (1999), no. 1 193–210.

[86] P. K. Agarwal and J. Pan, *Near-linear algorithms for geometric hitting sets and set covers*, in *Proceedings of 30th SoCG*, p. 271, ACM, 2014.

[87] H. Brönnimann and M. T. Goodrich, *Almost optimal set covers in finite vc-dimension*, *Discrete & Computational Geometry* **14** (1995), no. 4 463–479.

[88] N. H. Mustafa, R. Raman, and S. Ray, *Settling the apx-hardness status for geometric set cover*, in *Proceedings of 55th FOCS*, pp. 541–550, IEEE, 2014.

[89] C. Chekuri, K. L. Clarkson, and S. Har-Peled, *On the set multi-cover problem in geometric settings*, *ACM Transactions on Algorithms (TALG)* **9** (2012), no. 1 9.

[90] M. Cygan, F. Grandoni, S. Leonardi, M. Mucha, M. Pilipczuk, and P. Sankowski, *Approximation algorithms for union and intersection covering problems*, in *Proceedings of 31st FSTTCS*, p. 28, 2011.

[91] E. Eiben, J. Gemmell, I. Kanj, and A. Youngdahl, *Improved results for minimum constraint removal*, in *Proceedings of 32nd AAAI Conference on Artificial Intelligence*, 2018.

[92] M. Korman, M. Löffler, R. I. Silveira, and D. Strash, *On the complexity of barrier resilience for fat regions and bounded ply*, *Comput. Geom.* **72** (2018) 34–51.