
Algorithms for Obnoxious Facility Dispersion Problems in the Plane

THESIS

*Submitted in partial fulfillment
of the requirements for the degree of*

DOCTOR OF PHILOSOPHY

By

**S Vishwanath Reddy
ID No. 2019PHXF0420H**

Under the supervision of

Dr. Manjanna B



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

2024

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

Certificate

This is to certify that the thesis titled “**Algorithms for Obnoxious Facility Dispersion Problems in the Plane**” submitted by **S Vishwanath Reddy** ID No. **2019PHXF0420H** for the award of Ph.D. of the institute embodies original work done by him under my supervision.

Manjanna B

Supervisor

Dr. Manjanna B

Assistant Professor,

Department of Computer Science and Engineering,
National Institute of Technology Karnataka (NITK),
Surathkal, Mangalore.

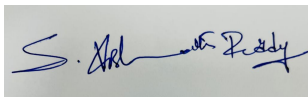
Place: BITS Pilani, Hyderabad Campus

Date: June 29, 2024

Declaration of Authorship

I, **S Vishwanath Reddy**, declare that this THESIS titled, ‘Algorithms for Obnoxious Facility Dispersion Problems in the Plane’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.



Student's signature

Date: July 04, 2024

Acknowledgements

I am deeply grateful to my supervisor, Dr. Manjanna B, for his invaluable support and mentorship throughout my doctoral journey. He introduced me to the fascinating field of Computational Geometry, and his guidance has been instrumental in shaping my research path. Throughout the entirety of my doctoral journey, Dr. Manjanna B has provided unwavering support, offering essential guidance, encouragement, and invaluable advice. I am truly grateful for his patient reading of my work, constructive feedback, and engaging conversations, all of which have greatly contributed to my personal and academic growth. Furthermore, Dr. Manjanna B consistently guided me with patience, care, and genuine concern whenever I encountered difficulties. His constant support has been crucial in helping me navigate the challenges I faced during my doctoral program. I am sincerely grateful for his academic and personal guidance and extend my heartfelt thanks to him.

I express my heartfelt gratitude for the invaluable experience of undertaking my Ph.D. at Birla Institute of Technology and Science, Pilani - Hyderabad campus. This journey has been truly unforgettable, filled with immense learning and knowledge acquisition. I am deeply aware that this accomplishment would not have been possible without the unwavering support and guidance bestowed upon me by numerous individuals. I consider it an immense privilege to have spent these four years in the esteemed Computer Science and Information Systems Department at Birla Institute of Technology and Science, Pilani - Hyderabad campus.

I express my gratitude to my DAC members, Prof. Paresh Saxena and Prof. Raghunath Reddy M, for their insightful comments and suggestions that significantly enhanced this research.

I am thankful to Prof. N. L. Bhanu Murthy, the Head of the Department of Computer Science and Information Systems, for providing all the necessary assistance during my thesis work.

I sincerely thank Prof. Tathagata Ray, the Convener of the Doctoral Research Committee in the Department of Computer Science and Information Systems, for his constant support.

I would like to thank Prof. G. Geethakumari, the former HoD of Computer Science and Information Systems, for her support.

I would also like to express my gratitude to the Department of Computer Science and Information Systems faculty members for their suggestions and encouragement during various progress seminar presentations and whenever I interacted with them.

I express my gratitude to Prof. Soumyo Mukherji, the director of BITS Pilani, Hyderabad Campus. Additionally, I would like to extend my thanks to Prof. Aivelu Manga Parimi, the Associate Dean of the Academic-Graduate Studies and Research Division, and his team for their continuous support throughout my Ph.D. journey.

I would like to take this opportunity to express my gratitude to the administrative and non-teaching staff of our department for their assistance in all official procedures, making my journey hassle-free.

I would like to thank Prof. Joseph S. B. Mitchell from Stony Brook University and Prof. N. R. Aravind from the Indian Institute of Technology Hyderabad for their valuable suggestions and collaborative work.

I would like to express my appreciation and gratitude to my seniors Gourish, Rajitha, Anirudh, Sahithi, Naresh, Priyanka, Shravan Kumar, Kavya, Deepa, as well as my colleagues Anil Kumar, Simran, Priyanka, Lalitha, Nida, Anand Agarwal, and my juniors for their support and encouragement.

I must thank my parents for their relentless efforts to educate me. Without them, I would not have reached where I am today.

Lastly, I would like to express my heartfelt appreciation to my wife, Navya, who has always cared for me, and my daughter Yokshitha and supported me through both challenging and joyful times.

S Vishwanath Reddy

Abstract

Motivated by operations research applications, which involve optimizing the placement of obnoxious facilities such as airports, industrial facilities, landfills, prisons, and other facilities that may impact each other and nearby residents, extensive research has been conducted on geometric dispersion problems within the field of computational geometry. This thesis investigates geometric dispersion and obnoxious facility location problems in the Euclidean plane and proposes exact, parameterized, and approximation algorithms.

Firstly, we study the following obnoxious facility location problem in the plane: locate k new obnoxious facilities amidst n ordered demand points (existing facility sites) so that none of the existing facility sites are affected (i.e., no new facility overlaps with the existing facility). We study this problem in two restricted settings. The obnoxious facilities are constrained to be on (i) a given horizontal line segment \overline{pq} , and (ii) the boundary arc of a given disk \mathcal{C} . For the problem in (i), we give an $(1 - \epsilon)$ -approximation algorithm that runs in $O((n \log n + k) \log \frac{\|pq\|}{2^{(k-1)\epsilon}})$ time, where $\epsilon > 0$, $k \geq 1$ and $\|pq\|$ is the length of \overline{pq} . We also propose two exact polynomial-time algorithms using binary search and Megiddo's [53] parametric search techniques, respectively, that run in time $O((nk)^2)$ and $O((n \log n + k)^2)$, respectively. Using the improved parametric technique [16], we give an $O(n \log^2 n)$ -time algorithm for $k = 2$. We also show that the $(1 - \epsilon)$ -approximation algorithm of (i) can be easily adapted to solve the problem (ii) with an extra multiplicative factor of n in the running time. Finally, we give a $O(n^3 k)$ -time dynamic programming solution to the *min-sum* obnoxious facility location problem restricted to a line segment where the demand points are associated with weights, and the goal is to minimize the sum of the weights of the points covered.

Next, we study a semi-obnoxious facility location problem constrained to a line in the plane. Suppose there exist two types of demand points that exhibit attraction and repulsion towards facilities. In that case, the problem is formulated as follows: we are given a set \mathcal{B} of blue points and a set \mathcal{R} of red points, all lying above a horizontal line ℓ , in the plane. Let the weight of a given point $p_i \in \mathcal{B} \cup \mathcal{R}$ be $w_i > 0$ if $p_i \in \mathcal{B}$ and $w_i < 0$ if $p_i \in \mathcal{R}$. Let $n = |\mathcal{B} \cup \mathcal{R}|$ and $d^0 (= d \setminus \partial d)$ be the interior of any geometric object d . We wish to pack k non-overlapping congruent disks d_1, d_2, \dots, d_k of minimum radius, centered on ℓ such that $\sum_{j=1}^k \sum_{\{i: \exists p_i \in \mathcal{R}, p_i \in d_j^0\}} w_i + \sum_{j=1}^k \sum_{\{i: \exists p_i \in \mathcal{B}, p_i \in d_j\}} w_i$ is maximized, i.e., the sum of the weights of the points covered by $\bigcup_{j=1}^k d_j$ is maximized. Here, the disks are the obnoxious or undesirable facilities generating nuisance or damage (with quantity equal to w_i) to every demand point (e.g., population center) $p_i \in \mathcal{R}$ lying in their interior. In contrast, they are the desirable facilities giving service (equal to w_i) to every demand point $p_i \in \mathcal{B}$ covered by them. The line ℓ represents a straight highway or railway line. These k semi-obnoxious facilities need to be established on ℓ to receive the largest possible overall service for the nearby

attractive demand points while causing minimum damage to the nearby repelling demand points. We show that the problem can be solved optimally in $O(n^4k^2)$ time. Subsequently, we improve the running time to $O(n^3k \cdot \max(n, k))$. Furthermore, we considered two special cases of the problem where points do not have arbitrary weights. In the first case, the objective is to cover the maximum number of blue points while avoiding red points. The second case aims to cover all the blue points with the minimum number of red points covered. We show that these two special cases can be solved in $O(n^3k \cdot \max(\log n, k))$ time. For the first case, when $k = 1$, we also provide an algorithm that solves the problem in $O(n^3)$ time, and subsequently, we improve this result to $O(n^2 \log n)$. The above-weighted variation of locating k semi-obnoxious facilities may generalize the problem that Bereg et al. [10] studied where $k = 1$, i.e., the smallest radius maximum weight circle is to be centered on a line. Furthermore, we consider a generalization of the weighted problem where we are given t horizontal lines instead of one line. We give an $O(n^4k^2t^5)$ time algorithm for this problem. Finally, we consider a discrete variant where a set of s candidate sites (in convex position) for placing k facilities is pre-given ($k < s$). We propose an algorithm that runs in $O(n^2s^2 + ns^5k^2)$ time for this discrete variant.

Next, we study a discrete variant of the obnoxious facility location problem where the candidate facility locations are positioned convexly and have no demand points. This particular variant of the problem is referred to as the k -dispersion problem on a convex set of points. It has been formulated as follows: given a set S of n points placed in convex position in the plane and an integer k ($0 < k < n$), the objective is to compute a subset $S' \subset S$ such that $|S'| = k$ and the minimum distance between a pair of points in S' is maximized. Based on the bounded search tree method, we propose an exact fixed-parameter algorithm in $O(2^k n^2 \log^2 n)$ time for this problem, where k is the parameter. The proposed exact algorithm improves on the algorithm of Akagi et al. [5], which requires time $n^{O(\sqrt{k})}$, whenever $k < c \log^2 n$ for some constant c . We then give an exact polynomial-time ($O(n^4k^2)$) algorithm for any $k > 0$, thus answering the open question about the complexity of this restricted dispersion problem. For $k = 3$, there is an $O(n^2)$ -time algorithm by Kobayashi et al. [44]. We then present a $O(n)$ -time $\frac{1}{2\sqrt{2}}$ -approximation algorithm for the problem when $k = 3$ if the points are given in convex position order.

Unlike the aforementioned facility location problems, the concept of a dominating set in graph theory is also employed for modeling facility location problems. The concept of a dominating set has received significant attention in graph theory as well as in computational geometry and has been extensively utilized by researchers as a modeling tool for various facility location problems. In our study, we focused on an edge-vertex dominating set problem, which is formulated as follows: given an undirected graph $G = (V, E)$, a vertex $v \in V$ is edge-vertex (ev) dominated by an edge $e \in E$ if v is either incident to e or incident to an adjacent edge of e . A set $S^{ev} \subseteq E$ is an edge-vertex dominating set (referred to as ev -dominating set and in short as EVDS) of G if every vertex of G is ev -dominated by at least one edge of S^{ev} . The minimum cardinality of an ev -dominating set is the ev -domination number. The edge-vertex dominating set problem

is to find a minimum *ev*-domination number. We prove that the *ev*-dominating set problem is **NP-hard** on unit disk graphs. We then prove that the EVDS problem admits a polynomial-time approximation scheme on unit disk graphs. At last, we also give a simple 5-factor linear-time approximation algorithm. Finally, we conclude the thesis and mention some open problems.

Contents

Certificate	i
Declaration of Authorship	ii
Acknowledgements	iii
Abstract	v
Contents	viii
List of Tables	x
List of Figures	xi
List of Algorithms	xiii
List of Abbreviations	xiv
List of Symbols	xv
1 Introduction	1
1.1 Objectives	7
1.2 Outline of the Thesis	8
2 Literature Review	11
2.1 Constrained Obnoxious Facility Location (COFL) problem and its variants	11
2.2 Semi-Obnoxious Facility Location (SOFL) problem	13
2.3 Dispersion problems	14
2.4 Edge Vertex Dominating Set problem	16
3 Constrained Obnoxious Facility Location on a Line Segment	17
3.1 Decision version of the COFL problem	18
3.2 FPTAS for the COFL problem	23
3.3 Polynomial time exact algorithms for the COFL problem	24
3.3.1 Algorithm based on exhaustive search	24
3.3.2 Algorithm based on parametric search	28
3.3.3 Improved algorithm for $k = 2$	31

3.4	Circular COFL problem	33
3.5	Min-sum Obnoxious Facility Location (MOFL) problem	35
3.5.1	Dynamic programming solution for any $k > 0$	36
3.6	Conclusion	38
4	Semi-Obnoxious Facility Location on a Line	39
4.1	Preliminaries	40
4.2	Computing the candidate radii	41
4.3	Transformation to the minimum weight k -link path problem	44
4.4	Special cases of CSOFL	50
4.4.1	The MAXBLUE-NORED problem for $k = 1$	52
4.5	SOFL on t -lines	54
4.6	Discrete SOFL with all facility sites in convex position	56
4.6.1	Dynamic programming algorithm	57
4.7	Conclusion	60
5	Max-Min k-Dispersion for Points in Convex Position in the Plane	61
5.1	Preliminaries	61
5.2	An exact fixed-parameter algorithm	62
5.2.1	Decision algorithm	63
5.2.2	The optimization scheme	64
5.3	An exact polynomial time algorithm	69
5.4	A linear time $1/2\sqrt{2}$ -approximation for $k = 3$	73
5.5	Conclusion	76
6	Edge-Vertex Domination in UDG	77
6.1	Hardness results	78
6.2	Polynomial time approximation scheme	82
6.2.1	Subset Construction	84
6.3	5-Factor approximation algorithm	87
6.4	Conclusion	89
7	Conclusion and Future Work	90
8	Summary	93
	Bibliography	95
	List of Publications	102
	Biographical Sketch	104

List of Tables

1.1	Summary of the proposed results	10
2.1	Summary of related work on dispersion problems	15

List of Figures

1.1	Solution to the COFL on a line for $k = 2$	2
1.2	EVDS (blue edges) vs PDS (green edges) in UDG.	6
1.3	EVDS (blue edges) vs TDS (red vertices) in UDG.	6
3.1	Region \mathcal{R} of distance L from \overline{pq}	19
3.2	Point p_i in the region \mathcal{R} has two points $c_{i,1}$ and $c_{i,2}$ on \overline{pq} at distance L	20
3.3	Another point p_j in the region \mathcal{R} has two points $c_{j,1}$ and $c_{j,2}$ on \overline{pq} at distance L	20
3.4	A point p_i is having two center points of \overline{pq} which are at unknown distance L	24
3.5	Left arc segment and right arc segment of disks d_j and $d_{j'}$	25
3.6	Illustration of case 2.	26
3.7	Two points p_i and $p_{i'}$ determining the radius r_{CAN}	27
3.8	Illustration of case 4(i).	28
3.9	Illustration of case 4(ii).	28
3.10	\mathcal{C}_1 and \mathcal{C}_2 which are at distance L from \mathcal{C}	34
3.11	An illustration of splitting of mega-intervals into mini-intervals and their weight calculations.	36
4.1	Configuration-0.	41
4.2	Configuration-1.	42
4.3	Illustration of calculating r_{CAN}	43
4.4	An optimal packing of 3 disks for a candidate radius λ , d_i is centered at an endpoint of the influence interval due to p_i	45
4.5	Adding of extra-points including s and t	45
4.6	Calculation of the weight of an edge.	45
4.7	Any four vertices of G' that satisfy the concave Monge property.	46
4.8	The optimal solution with only one blue point for $k = 1$	53
4.9	Candidate locations corresponding to the endpoint of the infeasible region p_{ℓ_i} and candidate radius λ	55
4.10	The blue point p_i lying on the boundary of d_3 will determine the radius.	57
4.11	Illustration of the candidate facilities will not determine the radius of disks in the optimal packing.	57
5.1	(a) Disks placed by the decision algorithm for $k = 4$ (b) Corresponding 2-way search tree.	63
5.2	Preprocessing and computation of a center vertex for d_{j+1}	66
5.3	(a) Greedy 1 (b) Greedy 2 (c) Greedy 3 (d) 2-way search tree for $k = 12$	69
5.4	OPT using Delaunay triangulation when $k = 6$	71
5.5	Extending Voronoi diagram to next Voronoi center v' by picking vertex $p_{\ell'}$	72

5.6	(a) e is the farthest point from the line segment \overline{ac} . (b) f is the nearest point to the perpendicular bisector \overline{pq}	74
5.7	The disks are centered at extreme points a, b and c	74
5.8	(a) When $r_1 > \max(\{r_{l1}, r_{l2}\}_{l=1}^6 \setminus \{r_1\})$. (b) When $r_{lm*} = \max\{r_{l1}, r_{l2}\}_{l=1}^6$ and $r_{lm*} > r_1$	76
6.1	(a) A planar graph G with max degree 3, and (b) The embedding of G on a grid.	79
6.2	A unit disk graph construction from embedding.	80
6.3	(a) Vertex cover of G . (b) EVDS of UDG.	81
6.4	4-Separable collection of edge sets $S = \{S_1, S_2, S_3, S_4, S_5\}$	84
6.5	EVDS = $\{e_1, e_2, e_3, e_4, e_5\}$, and minimum EVDS= $\{e\}$	88
6.6	Four adjacent Mega-cells.	89

List of Algorithms

1	$\text{DCOFL}(P, k, L)$	21
2	$\text{Exhaustive_Search}(P, k, \overline{pq})$	29
3	$\text{PARALLEL_DECISION}(P, 2, L)$	32
4	Exact-fixed-parameter.	65
5	Edge-vertex domination.	88

List of Abbreviations

OFL	O bnnoxious F acility L ocation
COFL	C onstrained O bnnoxious F acility L ocation on a line-segment
CCOFL	C ircular C onstrained O bnnoxious F acility L ocation
MOFL	M in-sum O bnnoxious F acility L ocation on a line-segment
SOFL	S emi O bnnoxious F acility L ocation
CSOFL	C onstrained S emi O bnnoxious F acility L ocation on a line
DKCONP	D iscrete k -dispersion on a C onvex P olygon
PTAS	P olynomial T ime A pproximation S cheme
FPTAS	F ully P olynomial T ime A pproximation S cheme
UDG	U nit D isk G raph
EVDS	E dge V ertex D ominating S et
PDS	P aired D ominating S et
TDS	T otal D ominating S et
MDS	M inimum D ominating S et
DAG	D irected A cyclic G raph
VD	V oronoi D iagram
FVD	F arthest point V oronoi D iagram
DT	D elaunay T riangulation
FPT	F ixed P arameter T ractable
OPT	O PTimal solution

List of Symbols

$\ \overline{pq}\ $	–	Length of a line-segment \overline{pq}
$(x(p_i), y(p_i))$	–	The coordinates of a point p_i
∂d	–	The boundary of a disk d
d^0	–	The interior of a disk d , i.e., $d^0 = d \setminus \partial d$
$\partial \mathcal{P}$	–	The boundary of a convex polygon \mathcal{P}
$dist(u, v)$	–	The minimum Euclidean distance between two points u and v
L_{arc}^j	–	The top-left arc segment of a disk d_j
R_{arc}^j	–	The top-right arc segment of a disk d_j
$L_{\overline{p_i p_j}}^-$	–	The left half-plane of line $\overline{p_i p_j}$
$\mathcal{E}_c(p_i, p_j, p_k)$	–	The circle circumscribing the three points p_i , p_j and p_k
$N_e(e')$	–	The edge neighborhood of an edge e'
$N_e[e']$	–	The closed edge neighborhood of an edge e'
$\Pi(i, j)$	–	The path from node i to node j
$\Pi_k(i, j)$	–	The k -link path from node i to node j
$w(i, j)$	–	The weight of an edge \overline{ij}
$w(d_j)$	–	The total weight of points covered by a disk d_j
\mathcal{L}_{CAN}	–	The set of all candidate radii
r_{CAN}	–	The candidate radius
r_c	–	The radius of the circle \mathcal{C}
$\mathcal{C}(d_j)$	–	The center of a disk d_j
$[n]$	–	$\{1, 2, \dots, n\}$

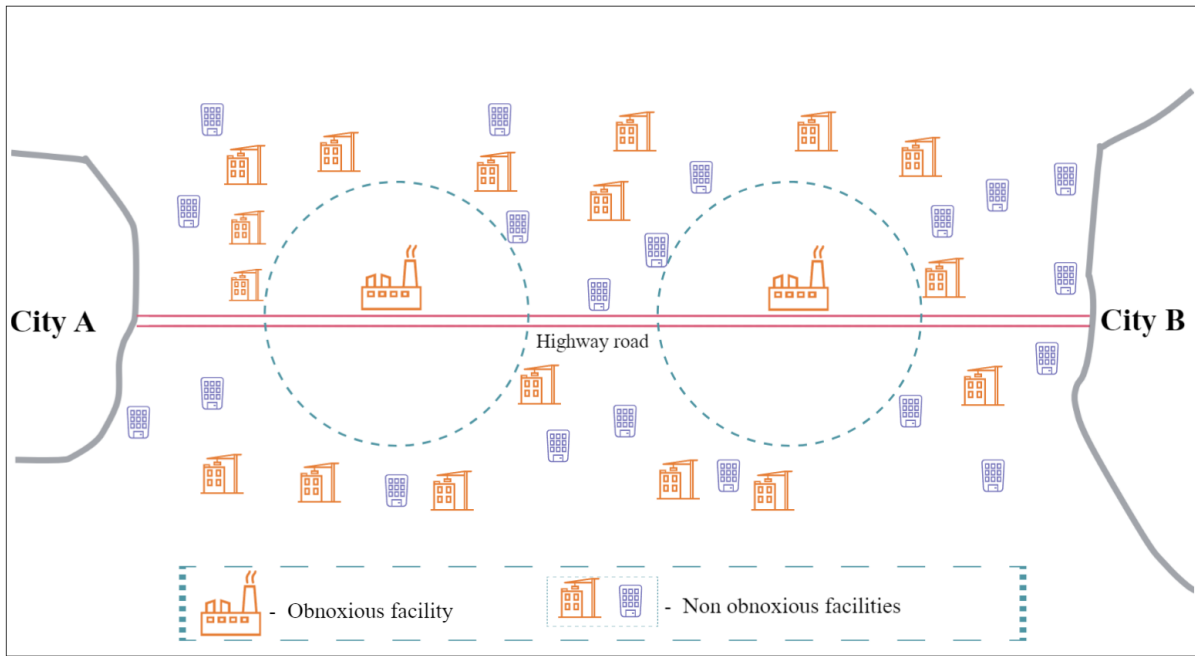
Chapter 1

Introduction

The study of facility location problems, also known as location theory, is a sub-field of operations research and computational geometry. An obnoxious facility can be defined as one that harms surrounding communities; hence, it is preferred to have them dispersed as much as possible while being farther from all kinds of social communities (called demand points in the literature). Extensive research has been conducted within the field of computational geometry, driven by the motivation to optimize the placement of obnoxious facilities like airports, industrial facilities, landfills, prisons, and other establishments that have the potential to impact both one another and the nearby residents.

The present thesis delves into the exploration of diverse geometric dispersion and obnoxious facility location problems in the Euclidean plane. In order to tackle the above-mentioned issues, we put forth a range of solution approaches, including exact, parameterized, and approximation algorithms. Our aim is to provide algorithmic solutions that can effectively solve several variations of these problems, ultimately contributing to advancing research in this field.

The obnoxious facility location (OFL) is a well-known topic in the operations research community. In most spatial location problems, the facilities are located as close as possible to clients such that the clients will get service by traveling less distance. In the case of obnoxious facilities, these facilities have to be placed as far as possible away from the other communities (such as residential areas, hospitals, schools, etc.) to minimize the nuisance generated by these obnoxious facilities. In this context, we address an obnoxious facility location problem, the continuous obnoxious facility location on a line segment (COFL) problem, motivated by the following application. We wish to find locations for establishing k obnoxious or undesirable facilities (such as garbage dump yards, industries generating pollution, etc.) along a straight highway road such that the pairwise distance between these new facilities and the distance between each of the new facilities and other existing non-obnoxious facilities (such as hospitals, schools, etc.) are maximized (see Figure 1.1 for $k = 2$). The need for placing these facilities on the sides of highway roads is due to their heavy

FIGURE 1.1: Solution to the COFL on a line for $k = 2$

transportation requirements. Apart from this application, the following scenario in establishing a sensor network used for security purposes can also be modeled as a COFL problem. Suppose we are given a line segment \overline{pq} representing a continuous set of potential locations to place a fixed number of sensors. These sensors have to monitor suspicious users originating from the continuous region around \overline{pq} . However, there are also fair users (point set P) around \overline{pq} . Now we need to place these sensors on \overline{pq} and assign a maximum possible uniform range to the sensors without causing any interference between them, but at the same time, none of the fair users fall within the range of any of the sensors. The formal definition of the problem is below:

Problem 1.1. The *constrained obnoxious facility location* (COFL) problem: Given a set $P = \{p_1, p_2, p_3, \dots, p_n\}$ of n demand points in the plane, a line segment \overline{pq} and a positive integer k , pack k maximum-radius (non-overlapping) congruent disks $d_1, d_2, d_3, \dots, d_k$ centered on \overline{pq} such that no point of P lies inside any of these disks, where $p = (x(p), y(p))$, $q = (x(q), y(q))$, and $y(p) = y(q)$.

The Problem 1.1 can be modified to a circular version, where the obnoxious facilities are centered on the boundary of the given circle instead of the line segment as defined below:

Problem 1.2. The *circular constrained obnoxious facility location* (CCOFL) problem: Given a set $P = \{p_1, p_2, p_3, \dots, p_n\}$ of n demand points and a predetermined circle \mathcal{C} with radius r_c in the plane and a positive integer k , the problem is to locate k facility sites centered on \mathcal{C} such that the minimum of the smallest distance (in terms of Euclidean distance) between a demand point in P and its nearest facility site, and the smallest distance between facility sites (in terms of arc

length), is maximized. Note that the disks representing two consecutive facility sites centered on \mathcal{C} may overlap strictly inside \mathcal{C} .

We also study a *min-sum* obnoxious facility location problem on a line segment where the points are associated with weights. This model of OFL is motivated by the situation where we must place some obnoxious facilities near to a number of communities such as residential neighborhoods, schools, hospitals, etc., and these facilities affect their close neighborhood area. Now, we would like to place the facilities so that the total number of people affected across all close-by communities by these facilities is minimized. The problem is formally defined below.

Problem 1.3. The *min-sum obnoxious facility location* (MOFL) problem: Given a horizontal line segment \overline{pq} and a set P of n weighted points $\{p_1, p_2, \dots, p_n\}$ (communities) whose weights (representing number of people in each such community) are given by w_1, w_2, \dots, w_n respectively, in the plane, a positive integer k and a distance $\lambda > 0$, pack k (non-overlapping) disks $d_1, d_2, d_3, \dots, d_k$ of radius λ centered on \overline{pq} such that $\sum_{j=1}^k \sum_{\{i|p_i \in d_j\}} w_i$ is minimized, i.e., the sum of weights of the points covered by these non-overlapping disks is minimized.

Typically, facility location problems involve two types of facilities: desirable ones like hospitals, fire stations, and post offices that should be located as close as possible to demand points (population centers), and undesirable ones like chemical factories, nuclear plants, and dumping yards that should be located as far away as possible from demand points to minimize their negative impact. However, the semi-obnoxious facility location (SOFL) problems have the unified objective of optimizing both negative and positive impacts on the repelling and attractive demand sites, respectively. In SOFL problems, the aim is to locate facilities at an optimal distance from both attractive and repulsive demand points. This creates a bi-objective problem where two objectives must be balanced. For example, when building an airport, it should be located far enough from the city to avoid noise pollution but close enough to customers to minimize transportation costs. In this context, we formally define the following semi-obnoxious facility location problem constrained to a line.

Problem 1.4. The *constrained semi-obnoxious facility location* (CSOFL) problem: Given a set \mathcal{B} of blue points and a set \mathcal{R} of red points, where each point $p_i \in \mathcal{B}$ has a weight $w_i > 0$ and each point $p_i \in \mathcal{R}$ has a weight $w_i < 0$, and let $|\mathcal{B} \cup \mathcal{R}| = n$. Assume these points lie above a given horizontal line ℓ . The objective is to pack k non-overlapping congruent disks d_1, d_2, \dots, d_k of minimum radius, centered on ℓ , such that the sum of the weights of the points covered by $\bigcup_{j=1}^k d_j$ is maximized, i.e., $\sum_{j=1}^k \sum_{i \in [n]: \exists p_i \in \mathcal{R}, p_i \in d_j} w_i + \sum_{j=1}^k \sum_{i \in [n]: \exists p_i \in \mathcal{B}, p_i \in d_j} w_i$ is maximized.

In many variants of the clustering and facility location problems (viz. k -center, k -median, etc.) that are studied in the literature [23, 24], we are given a set of n points, and among them, we need to locate k facilities such that some objective function is minimized. In contrast, in the obnoxious

facility location problems, we need to maximize an objective function. In the literature, this wider class of facility location problems that aim to maximize some diversity measures are called dispersion problems. In the case of the *max-min* k -dispersion problem, we need to maximize the minimum distance between the selected k facilities. The applications of k -dispersion problems arise in many areas. Consider a specific application where the k -dispersion problem can be used in which the given points are in convex position, as discussed below. For example, consider a convex island where some oil storage plants are to be established on the shore for transport using ships. Moreover, these plants should be kept as far away as possible so that any accident in one plant should not affect the other. We can model this problem as the k -dispersion problem, in which the plants are placed on the island's boundary to maximize the distance between any pair of plants. We define the problem formally below.

Problem 1.5. Discrete k -dispersion on a convex polygon (DKCONP): Given a set S of n points placed in the plane in convex position (forming a convex polygon \mathcal{P}) and an integer k ($0 < k < n$), the objective is to compute a subset $S' \subset S$ such that $|S'| = k$ and the minimum distance between a pair of points in S' is maximized.

Observe that the k -dispersion problem (Problem 1.5) on the set S can be equally stated as packing k congruent disks of maximum radius, with their centers lying at the vertices of the convex polygon \mathcal{P} .

To study facility location problems in wireless networks, a commonly used model in the literature is a unit disk graph model. As well known, a given set D of n disks of unit diameter (hence called *unit disks*) induces a graph G , where the graph G is called a unit disk graph (UDG) $G = (V, E)$ and is an undirected graph such that (i) each vertex v in whose vertex set V corresponds to a disk $d_v \in D$ of unit diameter in the plane, (ii) each edge (u, v) in whose edge set E corresponds to a pair of mutually intersecting disks d_u and d_v in the plane.

The problem of finding a minimum dominating set in UDG possesses applications within facility location scenarios where the objective entails minimizing the number of facilities (e.g., gateways or sink nodes) required to ensure coverage of all points of interest (e.g., network nodes). Consider, for instance, a scenario where wireless access points need to be positioned in a specific region to guarantee complete coverage. By solving the dominating set problem within the UDG that represents the given area, the minimum number of access points necessary to cover all locations can be determined. Many of the dominating set problems in UDG are **NP-hard**. Efficient approximation algorithms have been developed to solve the dominating set problems in UDGs. These algorithms are designed to identify an optimal or nearly optimal solution by selecting a subset of nodes that form a minimum dominating set. Through the utilization of such algorithms, both researchers and practitioners can make informed decisions regarding facility placement, aiming to achieve maximum coverage or cost minimization across a range of real-world applications. In this context, we studied a variant of the dominating set in UDG as discussed below.

Given an undirected graph $G = (V, E)$, the *edge neighborhood* of an edge $e' \in E$ is the set of edges in E which share a common vertex $v \in V$ with e' , i.e., the set of all edges which are adjacent to e' . The set of these neighbors of e' is represented as the set $N_e(e') = \{f \in E \mid e' \text{ and } f \text{ share a common vertex } v \in V\}$. The *closed edge neighborhood* of e' is defined as $N_e[e'] = N_e(e') \cup \{e'\}$. The *edge neighborhood* of a set $S \subseteq E$ is $N_e(S) = \bigcup_{e' \in S} N_e(e')$. Similarly, the *closed edge neighborhood* of a set $S \subseteq E$ is $N_e[S] = \bigcup_{e' \in S} N_e[e'] \cup S$. The *edge neighborhood of neighborhood* of e' is $N_e^2(e') = N_e(N_e(e'))$. Similarly, the r -th edge neighborhood is $N_e^r(e') = N_e(N_e^{r-1}(e'))$ for an integer $r > 1$.

Given an undirected graph $G = (V, E)$, a vertex $v \in V$ is *ev (edge-vertex)-dominated* by an edge $e \in E$ if v is incident to e (i.e., an endpoint of e) or if v is incident to an adjacent edge of e . A set $S^{ev} \subseteq E$ is an *edge-vertex dominating set* (EVDS) (referred to as *ev-dominating set*) of G if every vertex of G is *ev-dominated* by at least one edge of S^{ev} (at least two edges for *double edge-vertex dominating set*). The minimum cardinality of an *ev-dominating set* is the *ev-domination number*, denoted by $\gamma_{ev}(G)$. A *paired-dominating set* (PDS) of a graph $G(V, E)$ with no isolated vertices is a dominating set $S^{pr} \subseteq V$ and a sub-graph induced by S^{pr} in G have a perfect matching. The minimum cardinality of a PDS of G is denoted as $\gamma_{pr}(G)$. Note that EVDS and PDS may be completely different subsets of edges in the same graph, and their cardinalities may always be equal (see Figure 1.2). In Figure 1.2, the blue colored edges represent an EVDS. However, it should be noted that the set of these blue-colored edges does not fulfill the criteria to be classified as a PDS; instead, the PDS could correspond to the set of green edges. Another similar model, called *total domination* in a graph, is defined in terms of only vertices instead of *edge-vertex*. A *total dominating set* (TDS) of a graph $G = (V, E)$ is a *dominating set* $S^d \subseteq V$ such that every vertex $v \in S^d$ is adjacent to some other vertex in S^d . The TDS problem is to find such a set S^d of minimum cardinality. The minimum cardinality of TDS is denoted by γ_t . We can also view a TDS in a graph as a minimum cardinality set of pairs of adjacent vertices (hence, as a set of edges induced on these vertices), where these pairs may share a common vertex. Therefore, a TDS is also an EVDS and vice versa. However, a minimum cardinality EVDS may not be a minimum cardinality TDS, i.e., the set of all vertices incident to edges of a minimum cardinality EVDS may not necessarily form a minimum cardinality TDS, and vice versa (see Figure 1.3). In Figure 1.3, the blue edges represent the minimum cardinality EVDS, while the red vertices represent the minimum cardinality TDS. Here we can observe that the cardinality of the vertices incident to blue edges exceeds the minimum cardinality TDS and the cardinality of the edges incident to the red vertices also exceeds the minimum cardinality EVDS.

It is important to note that in UDG also, a minimum cardinality EVDS may not be a minimum cardinality TDS (see Figure 1.3). Therefore, the study of EVDS in UDG holds significant value.

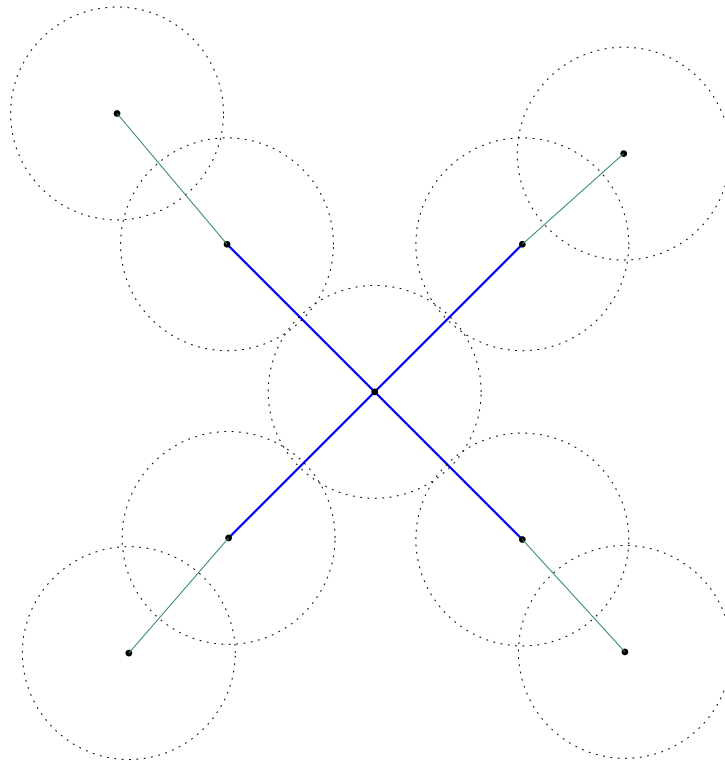


FIGURE 1.2: EVDS (blue edges) vs PDS (green edges) in UDG.

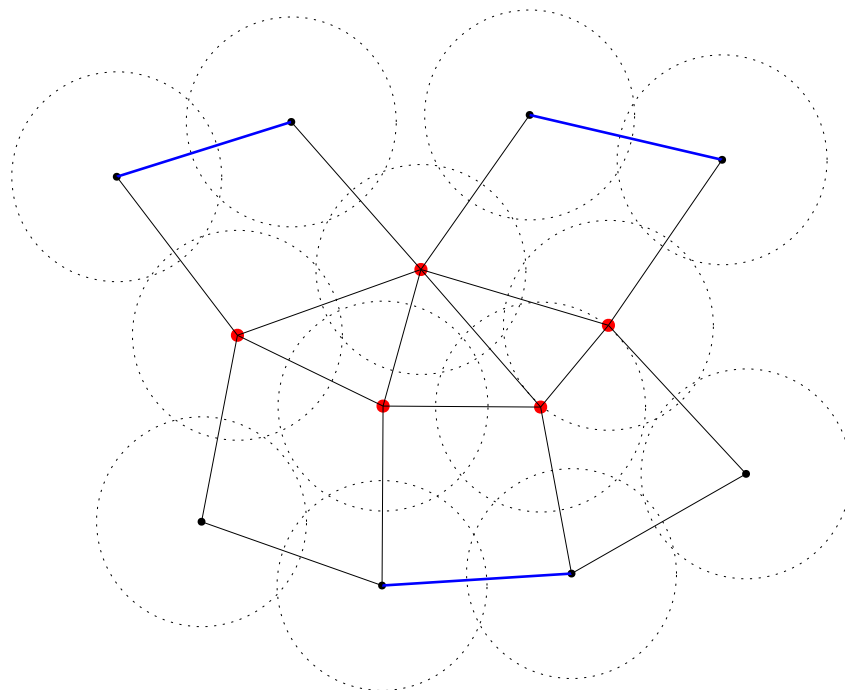


FIGURE 1.3: EVDS (blue edges) vs TDS (red vertices) in UDG.

The edge-vertex dominating set problem may have the following potential applications: In urban areas, certain facilities, such as parks or street parking zones, that has a significant impact over a wide area [48]. These facilities cannot be adequately represented or modeled by node centers alone because they are not solely accessible from a single entry point. Node centers typically

focus on capturing the accessibility of a location from a single point, which may not accurately reflect the spatial distribution and accessibility patterns of these facilities. However, edge centers offer a more faithful representation of such facilities as they consider the multiple entry points and capture the spatial extent and accessibility dynamics more comprehensively. By utilizing edge centers, we can better understand and model the true nature of these impactful urban facilities and their influence on the surrounding area.

An EVDS plays an important role in identifying vulnerable edges in a network [52]. These critical edges would give an attacker control over all connected nodes if compromised. EVDS helps prioritize security efforts by identifying the most critical edges. Security measures, like encryption or redundancy mechanisms, can then be directed toward securing these connections. Protecting these vital communication links ensures network integrity and minimizes the potential for data manipulation or interception by attackers.

A polynomial time algorithm \mathcal{A} of an optimization problem is an α -approximation algorithm if it generates a solution within a factor of α of the optimal solution (OPT), i.e., for maximization problems, it is $\frac{1}{\alpha} \cdot \text{OPT}$, and for minimization problems, it is $\alpha \cdot \text{OPT}$. For an α -approximation, we will call α as the *approximation ratio* or *approximation factor* of the algorithm.

An optimization problem admits a polynomial time approximation scheme (PTAS), if there is an algorithm \mathcal{A} that takes an instance I of size n and for each $\epsilon > 0$, that runs in a polynomial in n and gives a solution S_I which is either $(1 - \epsilon)$ -approximation or $(1 + \epsilon)$ -approximation to the optimal solution O_I for maximization problems ($S_I \geq (1 - \epsilon)O_I$) and minimization problems ($S_I \leq (1 + \epsilon)O_I$), respectively.

If the running time of algorithm \mathcal{A} is polynomial in the size of the problem and in $\frac{1}{\epsilon}$, then it is said that the optimization problem admits a fully polynomial time approximation scheme (FPTAS).

If a problem has some parameter k (where $k \in \mathbb{N}$), which is fixed as an input apart from n , then it is called parameterized problem. A parameterized problem is a fixed parameter tractable (FPT) if it can be solved by the parameterized algorithm for fixed k and has the running time of the form $O(f(k) \cdot n^{O(1)})$, where $f(k)$ is an arbitrary function dependent only on k .

1.1 Objectives

The above-defined problems (Problem 1.1, Problem 1.2, Problem 1.3, Problem 1.4 and Problem 1.5) are formulated to attain the following objectives of this research.

1. To study the problem of locating k obnoxious facilities on a given line segment and its variants.

2. To study the problem of locating k semi-obnoxious facilities on a given line and its variants.
3. To study the algorithmic complexity of the problem of packing k congruent disks centered at the vertices of a given convex polygon such that their common radius is maximized.
4. To check the hardness result of the edge-vertex dominating set problem on unit disk graphs. If it is **NP-hard**, then check whether it admits a polynomial time approximation scheme (PTAS) or not and design a better constant factor approximation algorithm.

1.2 Outline of the Thesis

Chapter 2: Literature Review. In this chapter, we will discuss the existing research on similar problems and compare our work with previous studies.

Chapter 3: Constrained Obnoxious Facility Location on a Line Segment. In this chapter, we first provide the formal definition of the decision version of the COFL problem, denoted as $\text{DCOFL}(P, k, L)$, and present a linear time algorithm based on a greedy approach for solving the decision version. Subsequently, we present an FPTAS (Fully Polynomial-Time Approximation Scheme) for the COFL problem, utilizing doubling search and bisection methods. This FPTAS involves multiple invocations of $\text{DCOFL}(P, k, L)$. Then, we discuss two exact algorithms for the COFL problem with polynomial time complexity. The first algorithm is based on the binary search and runs in $O((nk)^2)$ time. The second algorithm is based on the parametric search and runs in $O((n \log n + k)^2)$ time. Additionally, we introduce a faster algorithm based on the improved parametric search for $k = 2$, which achieves a time complexity of $O(n \log^2 n)$. Subsequently, we proceed to examine an FPTAS for the circular variant of the COFL problem, referred to as CCOFL. Finally, we present a dynamic programming-based polynomial time algorithm for solving the MOFL problem, which is a weighted min-sum version of the COFL. Then, we conclude the chapter.

Chapter 4: Semi-Obnoxious Facility Location on a Line. In this chapter, we first introduce and discuss various notations that will aid in comprehending the subsequent sections. Subsequently, we explore multiple configurations considered for computing the set of candidate radii, denoted as \mathcal{L}_{CAN} . Next, we delve into the transformation of the CSOFL problem into a minimum weight k -link path problem, given a candidate radius $\lambda \in \mathcal{L}_{\text{CAN}}$ that helps in solving the CSOFL problem exactly in $O(n^4 k^2)$ time. Additionally, we present an improved dynamic programming-based solution, which runs in $O(n^3 k \cdot \max(n, k))$ time. Subsequently, we examine two special cases of the CSOFL problem, namely the ALLBLUE-MINRED problem and the MAXBLUE-NORED problem. These cases involve only two sets of weighted points. We show that these problems can be solved in $O(n^3 k \cdot \max(\log n, k))$ time, and we also discuss the MAXBLUE-NORED problem for $k = 1$. Then, we extend the result of CSOFL to t -lines instead of a single

line. Finally, we discuss a discrete variant of SOFL when candidate facility sites are in convex position, and we conclude the chapter by giving final remarks.

Chapter 5: Max-Min k -Dispersion for the Points in Convex Position. In this chapter, first, we discuss various notations that will help in understanding subsequent sections in this chapter. We discuss an exact fixed-parameterized algorithm for DKCONP problem by defining and solving its decision version, identifying the set of candidate radii, and invoking the decision algorithm for candidate radii by doing a binary search on the set of candidate radii. Later, we present an exact polynomial-time algorithm with a time complexity of $O(n^4k^2)$. This algorithm utilizes two well-known concepts in computational geometry called Voronoi diagrams and Delaunay triangulation, which internally uses dynamic programming. Finally, we give a linear time $\frac{1}{2\sqrt{2}}$ -approximation for $k = 3$ by exploiting the elementary geometry properties, and then we conclude the chapter.

Chapter 6: Edge-Vertex Domination in UDG. In this chapter, first, we show that the decision version of the EVDS on UDG is NP-complete by describing a polynomial time reduction from the vertex cover problem, which is known to be NP-complete in planar graphs with maximum degree 3. Later, we show that the EVDS problem on UDG admits a PTAS, which utilizes the concept of an p -separated collection of subsets. Finally, we give a linear time 5-factor approximation algorithm, and then we conclude the chapter.

Chapter 7: Conclusion and Future Work. In this chapter, we present concluding remarks summarizing the key findings and contributions discussed in the preceding sections. Additionally, we identify several open problems that could serve as potential avenues for future research.

TABLE 1.1: Summary of the proposed results

Problem	Previous Results	Our Results
COFL	<p>The max-min OFL in rectilinear metric, constrained within a bounded region: For any $k > 0$, a $O(n^{k-2} \log^2 n)$ time algorithm exists [43].</p> <p>Euclidean metric: For $k = 2$, a $O(n \log n)$ time algorithm is reported in [43].</p>	<p>Euclidean metric constrained to a line segment: For any $k > 0$, we propose a (i) $(1 - \epsilon)$-approximation algorithm in $O((n \log n + k) \log \frac{\ pq\ }{2(k-1)\epsilon})$ time, for any $\epsilon > 0$ (FPT). (ii) Exact algorithm in $O(n^2 k^2)$ time. (iii) Also an improved exact algorithm in $O((n \log n + k)^2)$ time and finally, (iv) $(1 - \epsilon)$-approximation algorithm for circular variant of the problem (CCOFL).</p>
MOFL	<p>The min-sum OFL in rectilinear metric, within a bounded region: For $k = 1$, a $O(n \log n)$ time algorithm is presented [43].</p> <p>Euclidean metric: For $k = 1$, a $O(n^2)$ time algorithm is reported in [25].</p>	<p>MOFL in Euclidean metric, constrained to a line segment: For $k = 1$, a $O(n \log n)$ time algorithm is proposed, which improves the result in [25]. For any $k > 1$, a $O(n^3 k)$ time algorithm is proposed.</p>
CSOFL	<p>SOFL constrained to a line: when $k = 1$ there is a $O(n^2 \log n)$ time algorithm [10].</p>	<p>SOFL constrained to a line: We propose an algorithm for any k with time $O(n^4 k^2)$, which is then improved to $O(n^3 k \cdot \max(n, k))$. We presented a polynomial time algorithm for the same problem constrained to t-lines (instead of a single line) that run in $O(n^4 k^2 t^5)$ time. We also proposed a polynomial time algorithm for the discrete variant of the problem, where a set of s candidate sites are in convex position, and this algorithm runs in $O(n^2 s^2 + ns^5 k^2)$ time.</p>
DKCONP	<p>The generalized variant of DKCONP, where points need not be in convex position, is NP-complete [82], and it can be solved exactly in $n^{O(\sqrt{k})}$ time [5].</p> <p>When points are in convex position and $k = 3$, a $O(n^2)$ time algorithm is presented [44], recently.</p>	<p>When points are in convex position, we propose an $O(2^k n^2 \log^2 n)$ fpt-time algorithm and also proposed an exact algorithm in $O(n^4 k^2)$ time.</p>
EVDS	<p>The EVDS in bipartite graph [47] is NP-complete. Other variants of the EVDS, such as Roman dominating set [68], Liar's dominating set [41], and vertex-edge dominating set [42] in UDG are NP-complete.</p>	<p>We proved the EVDS in UDG is also NP-complete and showed that it admits a PTAS.</p>

Chapter 2

Literature Review

In this chapter, we survey all the work related to the problems investigated in this thesis. In Section 2.1, we discuss the related work of the constrained obnoxious facility location (COFL) problem and its variants. In Section 2.2, we survey the existing results of the semi-obnoxious facility location (SOFL) problem. Finally, in Section 2.3 and Section 2.4, we discuss the previous works on geometric dispersion problems and dominating set problems in UDG, respectively.

2.1 Constrained Obnoxious Facility Location (COFL) problem and its variants

The operations research and computational geometry communities have investigated several variants of obnoxious facility location problems. The obnoxious facility location problems can be modeled in many ways. Maximizing the cumulative minimum distance between obnoxious facilities and other non-obnoxious facilities in a given location is the most common way. Church and Garfinkel [15] introduced the obnoxious p -median problem of locating p facilities such that the cumulative minimum distance from non-obnoxious facilities to p obnoxious facilities is maximized. The obnoxious p -median problem is modeled as p -max-sum problem and is proved NP-hard [76]. Drezner and Wesolowsky [25] first studied this problem that positions only a single facility by modeling this facility as a rectangle or a circle. Their algorithm runs in $O(n^2)$ time in both the rectangular and circular cases. Later, Katz et al. [43] studied two variants of k -obnoxious facility location problems (i) max-min k -facility location, in which the distance between any two facilities is at least some fixed value and the minimum distance between a facility and a demand point is to be maximized, and (ii) min-sum one-facility location problem in which the sum of weights of demand points lying within a given distance to the facility is to be minimized. In the rectilinear metric, they solved the first problem in $O(n \log^2 n)$ time for $k = 2$ or 3 , and for any $k \geq 4$ they gave an algorithm in $O(n^{(k-2)} \log^2 n)$ time. We remark that the COFL problem for

$k = 2$, under the ℓ_∞ norm, can be solved in $O(n \log n)$ by employing the optimization technique of Frederickson and Johnson [31] instead of doubling search and bisection methods because the problem will be a special case of the problem (i) in [43] where we need to place facilities on a given line segment. In the same paper, Katz et al. [43] solved the problem (i) in the Euclidean metric for $k = 2$ in $O(n \log n)$ time. For the second problem, their algorithm runs in $O(n \log n)$ time (rectilinear case), which improves the previous results $O(n^2)$ [25]. Again, Drezner and Wesolowsky [26] formulated another variant of the obnoxious facility location problem: locating an obnoxious facility that is as far as possible from the arcs and nodes of a network. They gave a $(1 - \epsilon)$ -approximation algorithm that runs in $O(a^3 \log(1/\epsilon))$ time for the weighted version of the problem, where a is the number of arcs in the network. Later, its running time was improved by Michael [66] with a slight modification of the problem by considering a rectilinear (grid city) network and reducing its running time to $O(a^2 \log n \log(1/\epsilon))$ time, where n is the number of nodes in the network. Qin et al. [62] studied a variant of k -obnoxious facility location problem in which the facilities are restricted to lie within a given convex polygonal domain. They proposed a 2-factor approximation algorithm based on a Voronoi diagram for this problem, and its running time is $O(kN \log N)$, where $N = n + m + k$, n denotes the number of demand points, m denotes the number of vertices of the polygonal domain, and k is the number of obnoxious facilities to be placed. Díaz-Báñez et al. [20] modeled obnoxious facility as an empty circular annulus whose width is to be maximized. This refers to a max-min facility location problem such that the facility is a circular ring of maximum width, where the width is the absolute difference between the radii of the two concentric circles. They solved the problem in $O(n^3 \log n)$ time, and if the inner circle contains a fixed number of points, then the problem can be solved in $O(n \log n)$ time. Maximum width empty annulus problems for axis parallel squares and rectangles can be solved in $O(n^3)$ and $O(n^2 \log n)$ time, respectively [8]. Abravaya and Segal [2] studied the problem of locating the maximum cardinality set of obnoxious facilities within a bounded rectangle in the Euclidean plane such that their pairwise distance is at least a given threshold. They proposed a 2-approximation algorithm and also a PTAS based on shifting strategy [37]. Agarwal et al. [3] showed the application of Megiddo's parametric search [53] method to solve several geometric optimization problems in the Euclidean plane, such as the big stick problem, the minimum width annulus problem, and the problem of finding largest mutual visible spheres. Colmenar et al. [17] gave an approximation algorithm for an obnoxious p -median problem on a general network using a heuristic method known as a greedy randomized adaptive search procedure. Gokalp [34] gave an iterative greedy algorithm that produces a high-quality solution within a short time. Drezner et al. [21] developed an efficient global optimization method known as "Big Arc Small Arc" to solve the Weber obnoxious facility location problem. They tested it with up to 10,000 demand points using Euclidean, Manhattan, and $\ell_{1.78}$ norms. Most of these problems were solved optimally within a few seconds. Drezner et al. [22] studied two problems related to placing a facility within a planar network. In the first problem, they assumed that the network's links generate a nuisance. In the second problem, the facility itself generates a nuisance. The objective

in both cases is to place the facility in a way that minimizes the nuisance. They proposed exact and approximate methods to solve these problems and tested their methods on networks with up to 40,000 links. A recent review of many variants of obnoxious facility location problems and specialized heuristics to solve them is given in [14].

2.2 Semi-Obnoxious Facility Location (SOFL) problem

The semi-obnoxious facility location problem can be modeled as Weber's problem [49], where the repulsive points are assigned with negative weights. This problem is solved by designing a branch and bound method with the help of rectangular subdivisions [49]. The problem of locating multiple capacitated semi-obnoxious facilities is solved using a bi-objective evolutionary strategy algorithm where the objective is to minimize both non-social and social costs [77]. The problem of locating a single semi-obnoxious facility within a bounded region is investigated through the construction of efficient sets, which are the endpoints of efficient segments (edges of the bounded region or edges of the Voronoi diagram) formulated based on their mathematical properties derived from the given set of points [54]. A bi-objective mixed integer linear programming formulation was introduced and applied to this semi-obnoxious facility location problem [19].

Golpayegani et al. [35] introduced a semi-obnoxious median line problem in the plane using Euclidean norm and proposed a particle swarm optimization algorithm. Later, Golpayegani et al. [36] proposed a particle swarm optimization that solved the rectilinear case of the semi-obnoxious median line problem. Recently, Gholami and Fathali [33] solved the circular semi-obnoxious facility location problem in the Euclidean plane using a cuckoo optimization algorithm which is known to be a meta-heuristic method. Wagner [81] gave duality results in his thesis for a non-convex single semi-obnoxious facility location problem in the Euclidean space. In this thesis, we studied the k obnoxious facility location problem restricted to a line segment [69, 71]. We initially proposed a $(1 - \epsilon)$ -approximation algorithm [69], and then two exact algorithms based on two different approaches. The algorithms run in $O((nk)^2)$ and $O((n \log n + k)^2)$ time, respectively, for any $k > 0$ and finally, we gave an $O(n \log^2 n)$ time algorithm for $k = 2$ [71]. We also examined the weighted variant of the problem (where the influencing range of obnoxious facilities is fixed and demand points are weighted). We gave a dynamic programming-based solution that runs in $O(n^3 k)$ time for this variant. Following the appearance of the above results in a conference paper [71], Zhang [83] refined our result to $O(nk\alpha(nk) \log^3 nk)$ by reducing the problem to the k -link shortest path problem on a complete, weighted directed acyclic graph whose edge weights satisfy the convex Monge property, where $\alpha(\cdot)$ refers to the inverse Ackermann function. Section 4.3 of this thesis follows a similar strategy of reducing the weighted CSOFL to the k -link path problem, but the edge weights satisfy the concave Monge property.

The SOFL problem is also closer to the class of geometric separability problems, where we need to separate the given two sets of points with a linear or non-linear boundary or surface in a high-dimensional space. Geometric separability is an important concept in machine learning and pattern recognition. It is used to determine whether a set of data can be classified into distinct categories (say, good and bad points type) using a specific algorithm or model. Then, given an arbitrary data point, we can predict whether this point is good or bad depending on which side of the separation boundary hyperplane it falls. O'Rourke et al. [59] gave a linear time algorithm based on linear programming to check whether a circular separation exists. They also showed that the smallest disk and the largest separating circle could be found in $O(n)$ and $O(n \log n)$ time, respectively. If a convex polygon with k sides separation exists, then for $k = \Theta(n)$, the lower bound for computing the minimum enclosing convex polygon with k sides is $\Omega(n \log n)$ [27] and can be solved in $O(nk)$ time. While the separability problem using a simple polygon [29] was shown to be NP-hard, Mitchell [56] gave $(\log n)$ -approximation algorithm for an arbitrary simple polygon. Recently, Abidha and Ashok [1] have explored the geometric separability problems by examining rectangular annuli with fixed (the axis-aligned) and arbitrary orientation, square annuli with a fixed orientation, and an orthogonal convex polygon. For rectangular annuli with a fixed orientation, they gave $O(n \log n)$ time algorithm. They gave $O(n^2 \log n)$ time algorithm for cases with arbitrary orientation. For a fixed square case, the running time of their algorithm is $O(n \log^2 n)$, while for the orthogonal convex polygonal cases, it is $O(n \log n)$ time.

2.3 Dispersion problems

The discrete k -dispersion problem for $k \geq 3$ is known to be NP-complete even when the triangle inequality is satisfied [28]. The Euclidean k -dispersion problem is proved NP-hard by Wang and Kuo [82]. Akagi et al. [5] gave an algorithm to solve the k -dispersion problem in the Euclidean plane exactly in $n^{O(\sqrt{k})}$ time. They also gave an $O(n)$ -time algorithm to solve the special cases of the problem in which the given points appear in order on a line or on the boundary of a circle. Later, Araki and Nakano [6] improved the running time of [5] to $O(\log n)$ for the line case. Ravi et al. [63] proved that for the max-min k -dispersion problem on an arbitrary weighted graph, we cannot give any constant factor approximation algorithm within polynomial time unless P=NP. If the triangle inequality is satisfied by the edge weights, then we cannot approximate the problem with a better factor than $\frac{1}{2}$ in polynomial time unless P=NP. They also gave a polynomial time $\frac{1}{2}$ -approximation algorithm for the problem in graph metric.

Horiyama et al. [39] solved the max-min 3-dispersion problem in $O(n)$ time in both L_1 and L_∞ metrics when the given points are in a 2-dimensional plane. They also designed an $O(n^2 \log n)$ time algorithm for the 3-dispersion problem in L_2 metric. The 1-dispersion problem is trivial when the points are in a convex position, and we can solve the 2-dispersion problem in $O(n \log n)$

time by computing the diameter of the convex polygon formed by these points [67]. Recently, Kobayashi et al. [44] gave $O(n^2)$ -time algorithm for the 3-dispersion problem on a convex polygon. Recently, Mishra et al. [55] also studied the k -dispersion problem on convex polygon and gave $O(n^3)$ time algorithm for $k = 4$. In the same paper [55], they also gave a 1.733-factor approximation algorithm that runs in $O(n^4)$ time. In the literature, to the best of our knowledge, the complexity of the k -dispersion problem on a convex polygon for any k has remained open; it is resolved in our work here. When the points are arbitrarily placed in the Euclidean plane, the current best approximation algorithm is still the $\frac{1}{2}$ -approximation algorithm proposed by Ravi et al. [63] for the metric case. Hence, also from the point of designing ρ -approximation algorithm for $\rho > \frac{1}{2}$, the problem is open. Other related results in the literature are the following. Baur and Fekete [9] studied the problem of maximizing the rectilinear distance between a selected set of n points within a polygon, and they showed that this problem could not be approximated within the factor $\frac{13}{14}$ unless $P=NP$. Fekete and Meijer [30] studied a variant of the discrete k -dispersion problem with an objective of maximizing the average rectilinear distance between k facilities in d -dimensional space. They solved the problem in linear time when k is fixed and gave a polynomial-time approximation scheme when k is part of the input.

TABLE 2.1: Summary of related work on dispersion problems

Result	Reference
Metric k -dispersion problem	
NP-complete	Erkut [28], 1990
$\frac{1}{2}$ -approximation algorithm	Ravi et al. [63], 1994
No $> \frac{1}{2}$ -factor approximation unless $P=NP$	Ravi et al. [63], 1994
Euclidean k -dispersion problem	
NP-complete	Wang and Kuo [82], 1988
Exact algorithm in $n^{O(\sqrt{k})}$ time	Akagi et al. [5], 2018
For $k = 3$, solved in $O(n^2 \log n)$ time	Horiyama et al. [39], 2021
$O(n)$ time algorithm in both L_1 and L_∞ metric	Horiyama et al. [39], 2021
Euclidean k -dispersion problem when the points are in convex position	
For $k = 3$, solved in $O(n^2)$ time	Kobayashi et al. [44], 2021
For $k = 4$, solved in $O(n^3)$ time	Mishra et al. [55], 2025
1.733-approximation algorithm in $O(n^4)$ time	Mishra et al. [55], 2025
Exactly in FPT-time ($O(2^k n^2 \log^2 n)$) for any k	This thesis
Exactly in poly-time ($O(n^4 k^2)$) for any k	This thesis

2.4 Edge Vertex Dominating Set problem

The *edge-vertex dominating set* and *vertex-edge dominating set* in a graph were introduced by Peters [61]. The edge-vertex dominating set and vertex-edge dominating set problems are NP-complete, even when restricted to bipartite graphs [47]. For every nontrivial tree \mathcal{T} , an upper bound on $\gamma_{ev}(\mathcal{T})$ is $(\gamma_t(\mathcal{T}) + s - 1)/2$ where s is the number of support vertices (the vertex adjacent to a leaf) [79]. The *total domination number* (γ_t) of a tree is equal to the *ev-domination number* (γ_{ev}) plus one [45]. The vertex-edge dominating set problem in UDG is NP-complete [42]. Also, in [42], a polynomial time approximation scheme (PTAS) is proposed. Finding γ_{ve} even in cubic planar graphs is NP-hard [84]. The vertex-edge domination problem can be solved in linear time on block graphs [60]. The same paper also shows that finding γ_{ve} in undirected path graphs is NP-complete. Given a connected graph G with n vertices where $n \geq 6$, then we have $\gamma_{ve}(G) \leq \lfloor \frac{n}{3} \rfloor$ [85]. Boutrig et al. [12] gave an upper bound for the independent ve-domination number in terms of the ve-domination number for connected $K_{1,k}$ -free graph with $k \geq 3$ and also gave an upper bound on the ve-domination number for connected C_5 -free graph.

The double vertex-edge domination was introduced by Krishnakumari et al. [46]. They showed that finding γ_{dve} in a bipartite graph is NP-complete and also proved that for every non-trivial connected graph G , $\gamma_{dve}(G) \geq \gamma_{ve}(G) + 1$, and $\gamma_{dve}(\mathcal{T}) = \gamma_{ve}(\mathcal{T}) + 1$ or $\gamma_{dve}(\mathcal{T}) = \gamma_{ve}(\mathcal{T}) + 2$ for any tree \mathcal{T} . Finding γ_{dve} in chordal graphs is NP-complete [80]. They gave a linear time algorithm to find γ_{dve} in proper interval graphs and also showed that finding γ_{dve} in general graphs with vertices having a degree at most five is APX-complete. The double version of edge-vertex domination was studied by Sahin and Sahin [65]. They also gave the relationship between γ_{dev} and γ_{dve} , γ_t , γ_{ev} for trees and graphs, and also gave formulas to determine the *double ev-domination number* of paths and cycles. Sahin and Sahin [64] proved that the total ev-dominating set problem is NP-hard for bipartite graphs. They also showed that $(n - l + 2s - 1)/2$ is the upper bound for γ_{ev}^t for a tree \mathcal{T} with order n , l leaves and s supporting vertices. To the best of our knowledge, in the literature, the ev-dominating set problem is not yet studied in the context of geometric intersection graphs.

Chapter 3

Constrained Obnoxious Facility Location on a Line Segment

In this chapter, we consider three variants of the constrained obnoxious facility location (COFL) problem as follows:

The *constrained obnoxious facility location* (COFL) problem is defined as follows: Given a set $P = \{p_1, p_2, p_3, \dots, p_n\}$ of n demand points in the plane, a line segment \overline{pq} and a positive integer k , pack k maximum-radius (non-overlapping) congruent disks $d_1, d_2, d_3, \dots, d_k$ centered on \overline{pq} such that no point of P lies inside any of these disks, where $p = (x(p), y(p))$, $q = (x(q), y(q))$, and $y(p) = y(q)$.

The *circular constrained obnoxious facility location* (CCOFL) problem is defined as follows: Given a set $P = \{p_1, p_2, p_3, \dots, p_n\}$ of n demand points and a predetermined circle \mathcal{C} with radius r_c in the plane and a positive integer k , the problem is to locate k facility sites centered on \mathcal{C} such that the minimum of the smallest distance (in terms of Euclidean distance) between a demand point in P and its nearest facility site, and the smallest distance between facility sites (in terms of arc length), is maximized. Note that the disks representing two consecutive facility sites centered on \mathcal{C} may overlap strictly inside \mathcal{C} .

The *min-sum obnoxious facility location* (MOFL) problem is defined as follows: Given a horizontal line segment \overline{pq} and a set P of n weighted points $\{p_1, p_2, \dots, p_n\}$ (communities) whose weights (representing number of people in each such community) are given by w_1, w_2, \dots, w_n respectively, in the plane, a positive integer k and a distance $\lambda > 0$, pack k (non-overlapping) disks $d_1, d_2, d_3, \dots, d_k$ of radius λ centered on \overline{pq} such that $\sum_{j=1}^k \sum_{\{i|p_i \in d_j\}} w_i$ is minimized, i.e., the sum of weights of the points covered by these non-overlapping disks is minimized.

In our approach to solve the first problem, we first solve the decision version of the COFL problem in $O(n \log n + k)$ time. Then, using this decision algorithm as a subroutine, we give an $(1 - \epsilon)$ -approximation algorithm (FPTAS) in $O((n \log n + k) \log \frac{\|pq\|}{2^{(k-1)\epsilon}})$ time for the COFL problem, where $\epsilon > 0$ and $\|pq\|$ is the length of the segment \overline{pq} . We subsequently show that it is, in fact, possible to solve the COFL problem exactly in polynomial time for a fixed k by presenting two polynomial-time exact algorithms. The first algorithm is based on an exhaustive search by explicitly computing all candidate radii L and runs in $O((nk)^2)$ time. The second algorithm is based on Megiddo's parametric search [53] and runs in $O((n \log n + k)^2)$ time. We then discuss an $O(n \log n)$ -time algorithm to solve the COFL problem for $k = 2$, which is faster than the previous two algorithms. For the circular version (CCOFL) of the problem, we give a $(1 - \epsilon)$ -approximation algorithm (FPTAS) by adapting the FPTAS of COFL problem, with an extra multiplicative factor of n in the running time. We also study another restricted version, the MOFL problem. For $k = 1$, we show that the MOFL problem can be solved in $O(n \log n)$ time, improving the previous best result of $O(n^2)$ [25], and for any $k > 0$ we give a dynamic programming solution that runs in $O(n^3 k)$ time.

Consider the following notations, let r_{max} be the radius of disks in the optimal packing, and $\|pq\|$ denotes the length of the line segment \overline{pq} . Now, we first consider the following obvious observation, then discuss how to solve the decision version of the COFL problem.

Observation 3.1. If k is the number of disks that need to be packed on the segment \overline{pq} and r_{max} is the radius of the optimal packing, then $r_{max} \leq (\frac{\|pq\|}{2^{(k-1)}})$.

The main challenge in the COFL problem is to determine the optimal locations for the centers of k -disks such that these disks are packed on a given line segment \overline{pq} . This is due to the fact that there are infinitely many possible centers for these disks on \overline{pq} , and we must ensure that none of the given demand points lie within the interior of any disks packed.

3.1 Decision version of the COFL problem

Given a horizontal segment \overline{pq} , without loss of generality, we can assume that all the points in P lie in the upper half-plane defined by a line through \overline{pq} . After fixing the segment \overline{pq} , we define the following decision version of the COFL problem as follows:

DCOFL(P, k, L): Given a set P of n points lying above a line through \overline{pq} , a real number L (where $0 < L \leq (\frac{\|pq\|}{2^{(k-1)}})$) and an integer k , can we pack k non-overlapping disks of radius L centered on \overline{pq} such that none of these disks contains a point of P in their interior?

The solution to the DCOFL(P, k, L) problem is as follows. First, we identify the "infeasible" intervals on the line segment \overline{pq} for a given radius L with respect to each point $p_i \in P$. These

intervals are termed infeasible because if a disk with radius L is centered anywhere within these intervals, at least one point $p_i \in P$ will lie strictly inside the disk. Next, we determine the feasible intervals on \overline{pq} based on the previously identified infeasible intervals. Finally, we check if we can greedily pack k disks with radius L centered at the leftmost points on the feasible regions, as described below.

Consider a region \mathcal{R} whose boundary is at a distance L from the line segment \overline{pq} (i.e., the Minkowski sum of \overline{pq} and a disk of radius L) (see Figure 3.1).

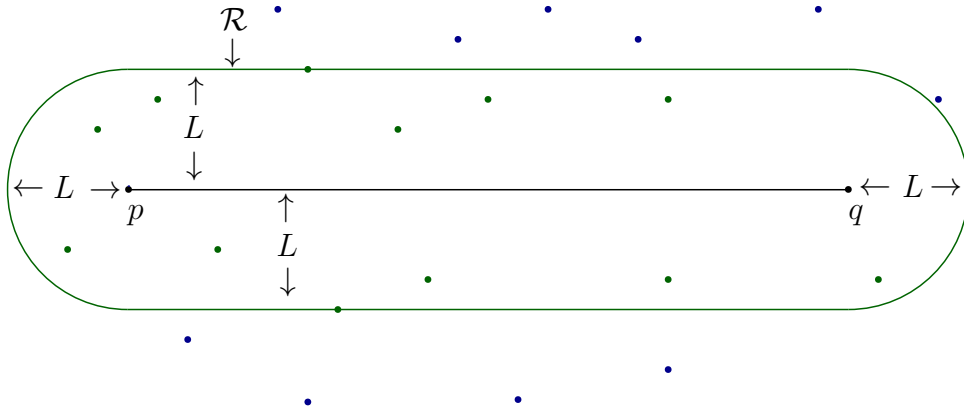


FIGURE 3.1: Region \mathcal{R} of distance L from \overline{pq} .

Observation 3.2. None of the points in P that lie outside the region \mathcal{R} will influence the choice of disk centers in the optimal solution to $\text{DCOFL}(P, k, L)$.

Now, from every point inside or on the boundary of the region \mathcal{R} , we can locate *center-points* on \overline{pq} which are at a distance L from this point, where a *center-point* is a candidate center point for the disks in a packing.

Lemma 3.3. *Each point in P that lies in the region \mathcal{R} will have at least one center point and at most two center points on \overline{pq} at a distance of L .*

Proof. Each of the points lying on the boundary of \mathcal{R} has exactly one center-point on \overline{pq} at distance L , whereas the points lying strictly in the interior of \mathcal{R} will have at most two center-points on \overline{pq} at distance L , (note that the coordinates of these center-points can be computed in $O(1)$ time using formulas from elementary geometry). Hence, there will be $O(1)$ center-points on the line segment \overline{pq} for every point in $P \cap \mathcal{R}$ (see Figure 3.2). \square

Observation 3.4. Let $p_i \in P$ be a point inside the region \mathcal{R} , and $c_{i,1}$ and $c_{i,2}$ be the center-points corresponding to p_i , then none of the k disks in an optimal solution to $\text{DCOFL}(P, k, L)$ will have their center points lying within the open interval $(c_{i,1}, c_{i,2})$ on the segment \overline{pq} .

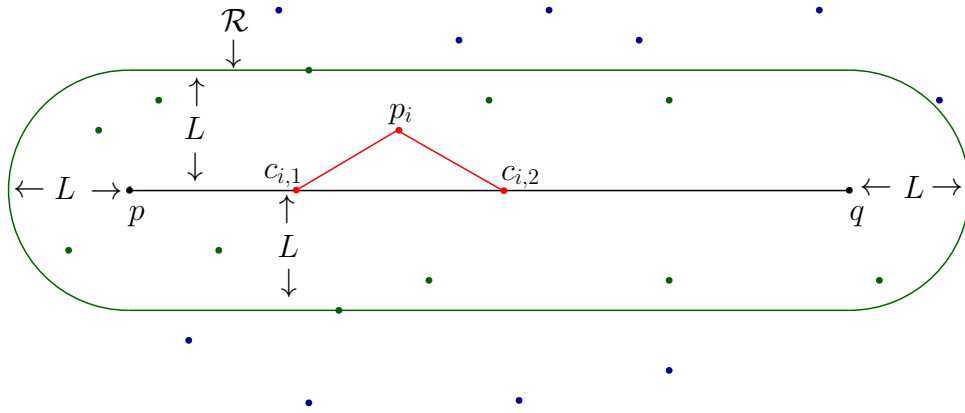


FIGURE 3.2: Point p_i in the region \mathcal{R} has two points $c_{i,1}$ and $c_{i,2}$ on \overline{pq} at distance L .

Now, consider another point $p_j \in P$ ($i \neq j$) inside the region \mathcal{R} , which has two center-points $c_{j,1}$ and $c_{j,2}$ on the segment \overline{pq} (see Figure 3.3). In figure 3.3, we can also observe that the intervals $[c_{i,1}, c_{i,2}]$ and $[c_{j,1}, c_{j,2}]$ formed by $\overline{c_{i,1}c_{i,2}}$ and $\overline{c_{j,1}c_{j,2}}$ are overlapping. Hence, from Observation 3.4 none of the k disks in the optimal solution will have their centers lying on the interval $([c_{i,1}, c_{i,2}] \cup [c_{j,1}, c_{j,2}]) \setminus \{c_{i,1}, c_{j,2}\}$, excluding the endpoints of the union of the two intervals.

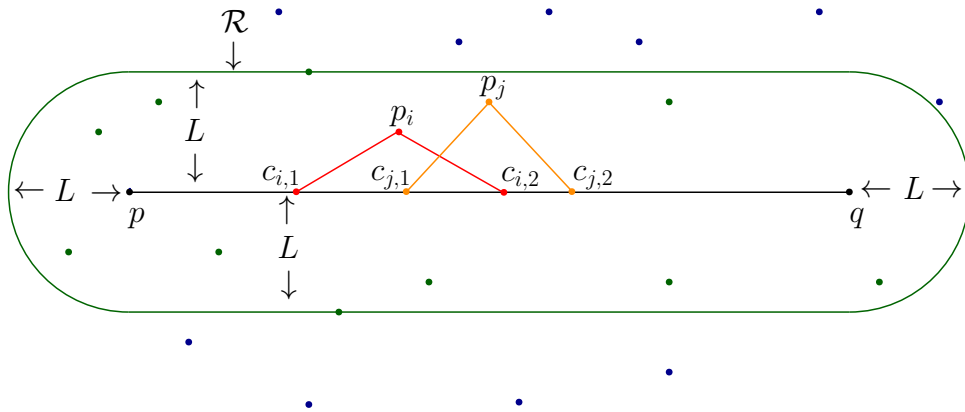


FIGURE 3.3: Another point p_j in the region \mathcal{R} has two points $c_{j,1}$ and $c_{j,2}$ on \overline{pq} at distance L .

Without loss of generality, let $\{p_1, p_2, \dots, p_m\}$ be the points of P lying strictly inside \mathcal{R} , above the line $y = y(p)$, and ordered from left to right, where $m \leq n$. We have seen that for every point p_i inside the region \mathcal{R} there will be two center-points on the line segment \overline{pq} which are at distance L , i.e., there is an interval $[l_i, r_i]$ for every point p_i inside the region \mathcal{R} , where $l_i = c_{i,1}$ and $r_i = c_{i,2}$. Merge all the overlapping intervals and then update the end points of the new intervals on \overline{pq} . Let I store these intervals after merging. Now, we will see how to compute I in $O(n \log n)$ time. First, list all the intervals $[l_i, r_i]$ in the increasing order of the x -coordinates of the left endpoint l_i . Consider the first interval in this ordered list and designate it as the current interval. Then, compare the right endpoint of the current interval with the left endpoint of the next interval in the ordered list. Merge them into a single interval if they overlap, e.g. $[c_{i1}, c_{i2}]$ and $[c_{j1}, c_{j2}]$ are merged into $[c_{i1}, c_{j2}]$ (see Figure 3.3). If they don't overlap, add the current interval (say, $[c_{i1}, c_{i2}]$) to I , and then make the next interval (i.e., $[c_{j1}, c_{j2}]$) as the current interval

and continue as above. We will repeat the above steps until either the last interval in the ordered list is merged into the current interval or the last interval is designated as the current interval and is subsequently added to I . Finally, we will have the set of $I = \{[l_1, r_1], [l_2, r_2], \dots, [l_{m'}, r_{m'}]\}$ as the resulting pairwise disjoint intervals, ordered by their left endpoints from left to right, where $m' \leq m$. This process will take $O(n \log n)$ time as we need to sort the original intervals.

Consider the complement of I with respect to \overline{pq} , denoted as $I^c = \{[p, l_1], [r_1, l_2], \dots, [r_{m'}, q]\}$, (here we assumed that none of the intervals in I contains the endpoints of \overline{pq}). Clearly, computing I^c will take $O(n)$ time. We can now tightly pack disks centered from point p to q on the segment \overline{pq} by placing them on the complemented intervals from I^c . For convenience assume $r_0 = p$, and $l_{m'+1} = q$. The k disks of radius L are packed on \overline{pq} by Algorithm 1.

Algorithm 1: DCOFL(P, k, L)

Compute I

Compute I^c where the intervals are ordered from p to q , and let $m = |I^c|$

if $|I^c| = \emptyset$ **then**

 | **return** (NO, \emptyset)

end

else

$j \leftarrow 0$

for each $i \leftarrow 1$ **to** m **do**

$\gamma \leftarrow \lfloor \frac{\text{length of } i\text{th interval of } I^c}{2L} \rfloor$

if $(j + \gamma + 1) \leq k$ **then**

 On the i th interval of I^c , pack the disks $d_{j+1}, d_{j+2}, \dots, d_{j+\gamma+1}$ of radius L .

 update $j \leftarrow (j + \gamma + 1)$

 update the intervals in I^c such that the distance between the left end point r_i of the left most interval $[r_i, l_{i+1}]$ in I^c and the right most point of ∂d_j on \overline{pq} is at least L .

if $j = k$ **then**

 | **break**

end

end

else

 On the i th interval of I^c , pack the disks $d_{j+1}, d_{j+2}, \dots, d_k$ of radius L .

 update $j \leftarrow k$

break

end

end

if $j = k$ **then**

 | **return** ($YES, \{d_1, d_2, \dots, d_k\}$)

end

else

 | **return** (NO, \emptyset)

end

end

Claim 3.5. Algorithm 1 (DCOFL(P, k, L)) correctly packs k disks of radius L if $L \leq r_{\max}$.

Proof. We prove the correctness of Algorithm 1 by induction on k , the number of disks to pack.

Base case ($k=1$): If $|I^c|$ is empty, then the algorithm returns NO and stops. If $|I^c|$ is not empty, this implies there is at least one interval in I^c on the line segment \overline{pq} such that if we place a disk d_1 of radius L centered on it, then d_1 does not contain any point from P in its interior.

Induction hypothesis: Assume for some integer k' , where $1 < k' < k$, Algorithm 1 successfully packed k' disks $d_1, d_2, \dots, d_{k'}$ with radius L as tightly as possible (the x -coordinates of their centers are as smallest as possible). We further assume that the first disk d_1 is centered at the left endpoint of the left-most interval in I^c .

Induction step: After packing k' disks with radius L :

- The first k' disks are placed as tightly as possible on the line segment \overline{pq} (the centers have the smallest possible x -coordinate).
- The leftmost disk is centered at the left endpoint of the left-most interval in I^c .
- If $L \leq r_{\max}$ and $k' + 1 \leq k$, then there always exists a point p on an interval in I^c . The point p is at a distance of at least $2L$ to the right of the center of $d_{k'}$ since the centers of $d_1, d_2, \dots, d_{k'}$, are lying at or to the left of the first k' disks respectively in any optimal packing. Now, we can center the $(k' + 1)^{\text{th}}$ at that point p .

Therefore, by induction, Algorithm 1 correctly packs k disks whenever $L \leq r_{\max}$. □

Theorem 3.6. *Algorithm 1 solves the DCOFL(P, k, L) problem in $O(n \log n + k)$ time.*

Proof. We analyze the time complexity of Algorithm 1 for solving the DCOFL problem, n denotes the number of points and k denotes the number of disks to pack.

- Given the segment \overline{pq} and L , the region \mathcal{R} can be computed in constant time.
- The points within the region \mathcal{R} can be found in time linear in $|P|$.
- Sorting the given points in P based on their x -coordinate and determining their corresponding intervals on \overline{pq} takes $O(n \log n)$ time. Construction of the sets I and I^c can be done in $O(n \log n)$ time.
- The packing of the k disks on the set I^c takes $O(n + k)$ time.
- Hence the total time is $O(n) + O(n \log n) + O(n + k) = O(n \log n + k)$.

□

3.2 FPTAS for the COFL problem

Now, we propose a fully polynomial time approximation scheme (FPTAS) for the *constrained obnoxious facility location* (COFL) problem, i.e., an $(1 - \epsilon)$ -approximation algorithm for any $1 > \epsilon > 0$.

In the COFL problem, our goal is to pack maximum-radius k (non-overlapping) congruent disks on the line segment \overline{pq} such that none of the points of the set P lie inside of any of these disks. As we discussed above, the answer to the $\text{DCOFL}(P, k, L)$ problem is YES if we are able to pack k congruent disks of radius L on \overline{pq} such that none of the points of P lie inside any of the disks, otherwise the answer is NO. Hence, to find the maximum radius of k congruent disks, we solve the $\text{DCOFL}(P, k, L)$ problem repeatedly for $L = 2^i$, where $i = 0, 1, 2, \dots$, as long as $\text{DCOFL}(P, k, L)$ problem returns YES. From Observation 3.1 the optimal radius r_{max} is at most $\frac{\|pq\|}{2^{(k-1)}}$. Let that small value of L be 2^j when the answer to the $\text{DCOFL}(P, k, L)$ problem is NO. However, when $L = 2^{j-1}$ the answer to $\text{DCOFL}(P, k, L)$ problem is YES. Hence, if $2^j \leq \frac{\|pq\|}{2^{(k-1)}}$ then r_{max} lies in the interval $[2^{j-1}, 2^j]$, else r_{max} lies in the interval $[2^{j-1}, \frac{\|pq\|}{2^{(k-1)}}]$ for a given set P of n points, a line segment \overline{pq} and an integer k . Given a real number ϵ , we bisect the interval $[2^{j-1}, 2^j] \log \frac{1}{\epsilon}$ time. Initially, $|2^j - 2^{j-1}| \leq r_{max}$, since we are able to pack k disks of radius 2^{j-1} . Let $[\alpha, \beta]$ be the interval after bisecting the interval $[2^{j-1}, 2^j]$ by $\log \frac{1}{\epsilon}$ times. Then $|\alpha - \beta| \leq \frac{|2^j - 2^{j-1}|}{2^{\log \frac{1}{\epsilon}}} \leq \frac{r_{max}}{2^{\log \frac{1}{\epsilon}}} \leq \epsilon r_{max}$. Hence, r_{max} lies in the interval $[\alpha, \beta]$, which implies that $\alpha \geq \beta - \epsilon r_{max} \geq (1 - \epsilon)r_{max}$. Therefore, the radius of the k congruent disks returned along with a positive answer by the last invocation of Algorithm 1 with $r = \alpha$ is at least $(1 - \epsilon)r_{max}$, where ϵ is an input parameter. Hence, we have the following theorem.

Theorem 3.7. *Given a line segment \overline{pq} and a set P of n points in the plane, we can get a $(1 - \epsilon)$ -factor approximation algorithm with $\epsilon > 0$ for the COFL problem, that runs in $O((n \log n + k) \log(\frac{\|pq\|}{2^{(k-1)\epsilon}}))$ time, by employing doubling search and bisection methods.*

Proof. From Theorem 3.6, we know that each call to solve the $\text{DCOFL}(P, k, L)$ problem will take $O(n \log n + k)$ time. Doubling search guarantees that the optimal radius r_{max} of k congruent disks lies either in the interval $[2^{j-1}, 2^j]$ or $[2^{j-1}, \frac{\|pq\|}{2^{(k-1)}}]$. We then divide this interval by $\log \frac{1}{\epsilon}$ times. In the worst case the number of invocations of the $\text{DCOFL}(P, k, L)$ problem is $O(\log \frac{\|pq\|}{2^{(k-1)}} + \log \frac{1}{\epsilon})$ time, for an input parameter ϵ . Hence, the total running time is $O((n \log n + k) \log(\frac{\|pq\|}{2^{(k-1)\epsilon}}))$. \square

Remark 3.8. The COFL problem for $k = 2$, under the ℓ_∞ norm, can be solved in $O(n \log n)$ by employing the optimization technique of Frederickson and Johnson [31] instead of doubling search and bisection methods because the problem will be a special case of the one in [43].

3.3 Polynomial time exact algorithms for the COFL problem

This section discusses two polynomial time exact algorithms for the COFL problem. First, we discuss a solution based on an exhaustive search¹ followed by a solution based on Meggido's parametric search [53]. Here, we first define a few more notations required to present our exact algorithms, as follows. Let $I = \{[x_{i,1}, x_{i,2}] | p_i \in P\}$ be the set of maximal intervals on \overline{pq} ordered from p to q (ordered by the x -coordinates $x(p_i)$ of $p_i \in P$), where every point in each interval $[x_{i,1}, x_{i,2}]$ is at distance at most L from $p_i \in P$ (see Figure 3.4). The decision algorithm returns YES if we can place k pairwise interior-disjoint disks of radius L , centered on \overline{pq} such that none of the disk centers lie in the interior of the intervals in I . Otherwise, the algorithm returns NO. Let $I^c = \{[x_{i,2}, x_{j,1}] | p_i, p_j \in P, i < j\} \cup \{[x_{0,2}, x_{1,1}], [x_{n,2}, x_{n+1,1}]\}$ be the set of complemented intervals of I , where every point in each interval $[x_{i,2}, x_{j,1}] \in I^c$ is at distance at least L from every point in P and $x_{0,2} = x(p)$, $x_{n+1,1} = x(q)$. Note that for every interval $[x_{i,2}, x_{j,1}] \in I^c$, no other interval of I lie entirely or partially between $x_{i,2}$ and $x_{j,1}$. Let \mathcal{R} be the Minkowski sum of \overline{pq} and a disk of radius L , i.e., the union of \overline{pq} and the region swept by a disk of radius L when its center moves along \overline{pq} .

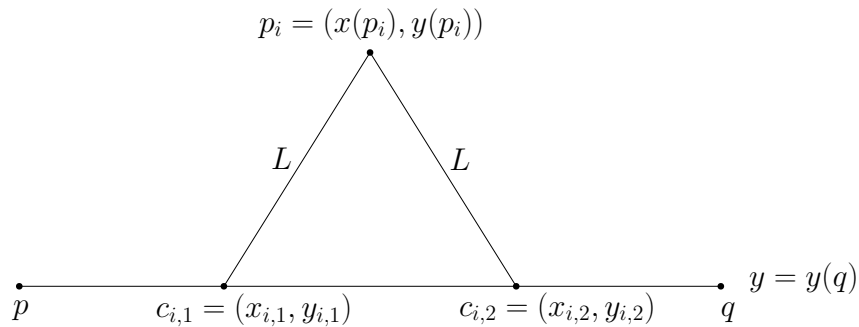


FIGURE 3.4: A point p_i is having two center points of \overline{pq} which are at unknown distance L .

3.3.1 Algorithm based on exhaustive search

The approach here is that we first compute the set \mathcal{L}_{CAN} of all candidate radii L , based on some observations of possible positions of all k disks in an optimal packing. We then repeatedly call the selection algorithm (e.g., see [18]) to find the median of \mathcal{L}_{CAN} . Then, we invoke the decision algorithm (Algorithm 1) each time we peek the median element by setting L to this element as the candidate radius. We continue this search until we find the radius L^* (maximum radius) by repeatedly removing one-half of the elements of \mathcal{L}_{CAN} based on the result returned by the decision algorithm until there is only one element in \mathcal{L}_{CAN} . Hence, $L^* = r_{\text{max}}$.

¹Here, exhaustive search involves considering all possible potential radii based on the positions of disks relative to the given points and \overline{pq} in an optimal packing and determining the one that results in maximum radius along with valid packing.

Consider an instance of the COFL problem, i.e., a line segment \overline{pq} , a set P of n points, and an integer k . As already assumed, without loss of generality, let all the points of P be lying above the line through the segment \overline{pq} . Now, we need to place k non-overlapping congruent disks of the maximum radius with centers lying on \overline{pq} without violating the constraint that no point of P lies inside any of these disks. The maximum radius of the disks depends on some “*influencing points*” in P for the given position of the fixed horizontal line segment \overline{pq} .

Definition 3.9. The influencing points $P^{inf} \subseteq P$ are those points which satisfy the following criteria: given a point $p_i \in P^{inf}$ (or a pair of points $p_i, p_{i'} \in P^{inf}$), we can place k pairwise disjoint congruent disks centered on \overline{pq} such that (i) p_i lies on the boundary of one of these disks (or p_i and $p_{i'}$ lie on the boundary of the same disk or on the boundaries of two different disks), and (ii) if p_i (or $p_{i'}$ or both) is (are) removed from P , then the radius of these disks may be increased while their centers are perturbed on \overline{pq} to keep their pairwise disjointness intact, and (iii) none of the other points of P lie in the interior of any of these disks before and after the radius increases.

Lemma 3.10. *Given that the position of \overline{pq} is fixed, for an optimal solution to the COFL problem in ℓ_2 metric, there are at most two points in $P^{inf} \subseteq P$ that will determine the optimal radius r_{max} of the disks in the corresponding packing, and also $|\mathcal{L}_{CAN}| = O(n^2k^2)$.*

Proof. The proof is based on case analysis. First, let $r_{CAN} \in \mathcal{L}_{CAN}$ be a candidate radius.

Case 1. The set P^{inf} is empty and then $r_{CAN} = \frac{\|pq\|}{2(k-1)}$.

If this case is not satisfied, then there will be at least one point lying on the boundary of some disk in the optimal packing, which will influence the value of r_{max} . Among these points in P lying on the boundaries of the optimal disks, let p_i be the leftmost, i.e., the point with the smallest x -coordinate. Let d_1, d_2, \dots, d_k be the disks ordered from left to right in the optimal packing. Now we will examine all possible positions of these disks for every subset of at most two influencing points in P^{inf} . We will also show how to compute the corresponding candidate radii $r_{CAN} \in \mathcal{L}_{CAN}$. To this end, consider a disk d centered on \overline{pq} . Divide the part of the boundary arc ∂d of d lying above \overline{pq} into left and right arc segments (denoted as L_{arc} and R_{arc} respectively) by a vertical line through the center of d (see Figure 3.5). We then show that at most two points $p_i, p_{i'} \in P^{inf} \subseteq P$ determine the radius r_{max} of the disks in an optimal packing. Let d_j and $d_{j'}$

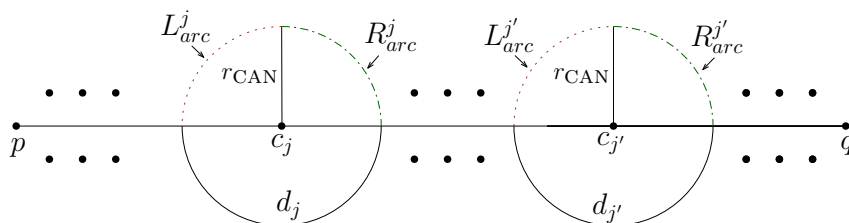


FIGURE 3.5: Left arc segment and right arc segment of disks d_j and $d_{j'}$.

(possibly $j = j'$) be the two disks, on whose boundary arc segments L_{arc}^j or R_{arc}^j , and $L_{arc}^{j'}$ or $R_{arc}^{j'}$ the two points p_i and $p_{i'}$ lie. Based on all possible positions of d_j and $d_{j'}$ for any pair p_i and $p_{i'}$, we have the following cases.

Case 2. Assume that p_i lies on R_{arc}^j of d_j such that its position determines the radius of the disk (there is no point $p_{i'}$ that influences the radius) (see Figure 3.6). For any $j = 1, 2, \dots, k$ in the optimal packing, p_i can lie on R_{arc}^j of d_j . For each such point $p_i \in P^{inf}$ the candidate radius r_{CAN} can be calculated and stored in \mathcal{L}_{CAN} . Hence the number of candidate radii in this case is $O(nk)$. The candidate radius r_{CAN} with respect to every point can be calculated using the below equation:

$$2(j-1)r_{CAN} = x(p_i) - \sqrt{r_{CAN}^2 - (y(p_i) - y(q))^2} - x(p) \quad (3.1)$$

The candidate radius $r_{CAN} \in \mathcal{L}_{CAN}$ is calculated from the above equation as we know the value of every term of the equation except r_{CAN} . The mirror case of this where the point $p_{i'}$ is the rightmost point and lies on $L_{arc}^{j'}$ of $d_{j'}$ can be handled similarly. In this case, r_{CAN} can be calculated using the below equation.

$$(2(k-j') + 1)r_{CAN} = \|pq\| - x(p_{i'}) + x(p) + r_{CAN} - \sqrt{r_{CAN}^2 - (y(p_{i'}) - y(q))^2} \quad (3.2)$$

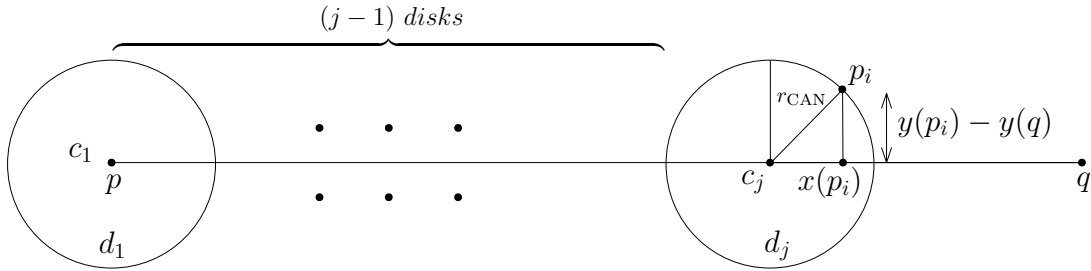


FIGURE 3.6: Illustration of case 2.

Case 3. Let both the points $p_i, p_{i'} \in P$ be determining the radius of the disks in the optimal packing. Then the point p_i can lie on L_{arc}^j or R_{arc}^j of d_j , where $j = 1, 2, \dots, k$ (see Figure 3.7). Similarly, the point $p_{i'}$ can also lie on $L_{arc}^{j'}$ or $R_{arc}^{j'}$ of $d_{j'}$, where $j' = j, j+1, j+2, \dots, k$. Here, the disks centered between p_i and $p_{i'}$ are compactly packed and determine the optimal radius along with the positions of p_i and $p_{i'}$. It is easy to observe that there are a constant number c of possible positions for the two disks d_j and $d_{j'}$ in an optimal solution such that the two points $p_i, p_{i'}$ lie on their boundaries while the radius r_{CAN} can not be increased by repositioning the centers of some or all k disks. Therefore, the number of all candidate radii r_{CAN} is $\binom{n}{2} \sum_{j=1}^k \sum_{j'=j}^k c = k^2 c \binom{n}{2} = O(cn^2k^2)$.

The candidate radii values $r_{\text{CAN}} \in \mathcal{L}_{\text{CAN}}$ can be computed from the following equation:

$$x(p_{i'}) - x(p_i) = \left[2(j' - j)r_{\text{CAN}} \pm_{(1)} \sqrt{r_{\text{CAN}}^2 - (y(p_i) - y(q))^2} \pm_{(2)} \sqrt{r_{\text{CAN}}^2 - (y(p_{i'}) - y(q))^2} \right] \quad (3.3)$$

wherein replacing $\pm_{(1)}$ with $+$ indicates that p_i lies on L_{arc}^j of d_j and with $-$ indicates that p_i lies on R_{arc}^j of d_j . Similarly, replacing $\pm_{(2)}$ with $+$ indicates $p_{i'}$ lies on $R_{\text{arc}}^{j'}$ of $d_{j'}$ and with $-$ indicates $p_{i'}$ lies on $L_{\text{arc}}^{j'}$ of $d_{j'}$.

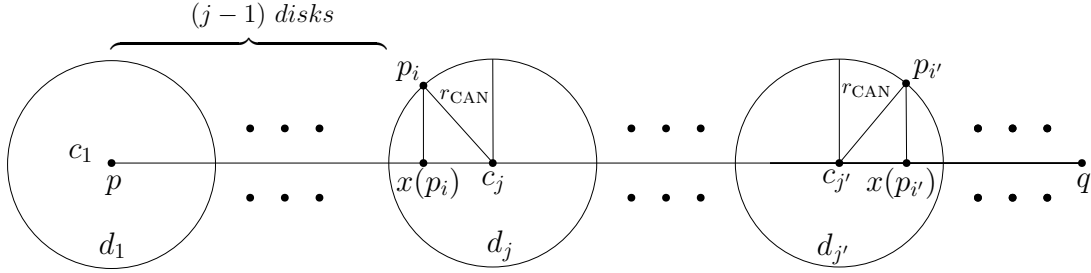


FIGURE 3.7: Two points p_i and $p_{i'}$ determining the radius r_{CAN} .

Case 4. Assume the point $p_i \in P$ lies on R_{arc}^j of some disk d_j and there are also other points $p_{i'}, p_{i''} \in P$ (optional) lying on boundary arcs of disks on the right of d_j . Based on all possible positions of $p_i, p_{i'}$ and $p_{i''}$ we can observe that at most two of these points will determine r_{CAN} , i.e., each of these possible positions corresponds to one of the above cases, as follows.

- (i) When $p_{i'} \in P$ lies on L_{arc}^{j+1} of the disk d_{j+1} and there is empty space between d_j and d_{j+1} (see Figure 3.8). This corresponds to *Case 2* above as either $j - 1$ disks on the left of p_i or $k - j - 1$ disks on the right of $p_{i'}$ are compactly packed with their radius increased to the maximum possible value.
- (ii) When $p_{i'} \in P$ lies on $L_{\text{arc}}^{j'}$ of the disk $d_{j'}$ and $p_{i''} \in P$ lies on the disk $d_{j''}$, where $j' - j > 1$ and $j'' - j' > 1$ (see Figure 3.9). Then, the disks in the optimal packing can be partitioned into four subsequences of consecutive disks $d_1, d_2, \dots, d_j, d_j, d_{j+1}, \dots, d_{j'}, d_{j'}, d_{j'+1}, \dots, d_{j''}$, and $d_{j''}, d_{j''+1}, \dots, d_k$. In at least one of these, the disks must be compactly packed with their radius increased to the maximum possible value; otherwise, it would contradict that we have the optimal packing (since their radius can be increased). Hence, one of the above cases applies, and at most two points among $p_i, p_{i'}, p_{i''}$ determine the value of r_{CAN} .

The constant $c = 10$ because of the following: (i) no point lies on the boundary of any disk in optimal packing, (ii) p_i lies on L_{arc}^j or R_{arc}^j , (iii) both $p_i, p_{i'}$ lie on L_{arc}^j or R_{arc}^j , (iv) p_i lies on L_{arc}^j and $p_{i'}$ lies on R_{arc}^j , (v) p_i lies on R_{arc}^j and $p_{i'}$ lies on R_{arc}^j or p_i lies on R_{arc}^j and $p_{i'}$ lies on L_{arc}^j or p_i lies on L_{arc}^j and $p_{i'}$ lies on R_{arc}^j or p_i lies on L_{arc}^j and $p_{i'}$ lies on L_{arc}^j . Observe that (i) corresponds to *Case 1* and contributes 1, (ii) corresponds to *Case 2* and contributes 2, (iii)

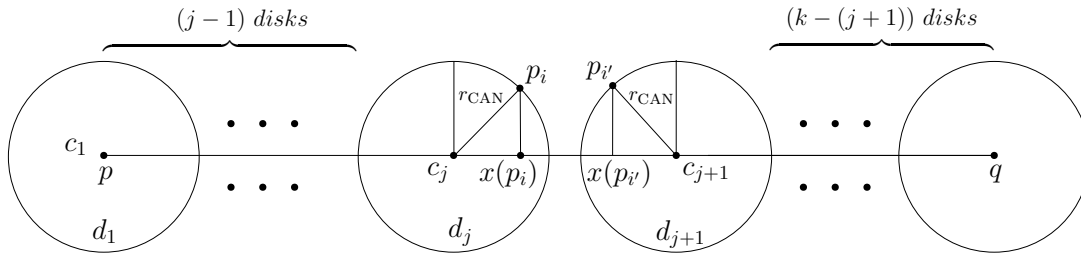


FIGURE 3.8: Illustration of case 4(i).

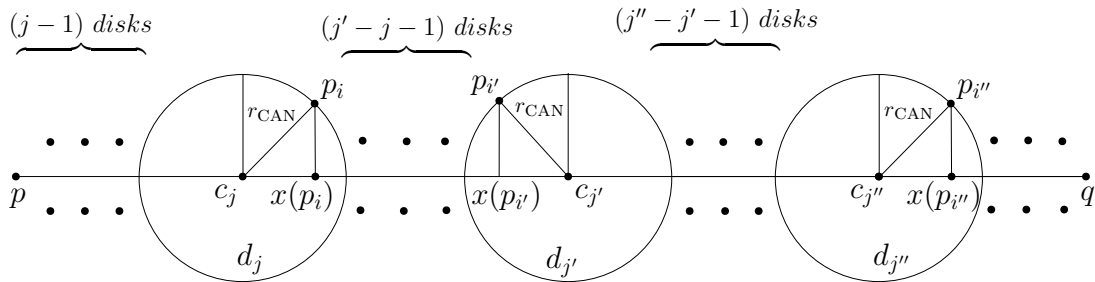


FIGURE 3.9: Illustration of case 4(ii).

contributes 2, (iv) contributes 1, (v) contributes 4, hence $c = 1 + 2 + 2 + 1 + 4 = 10$. Therefore, $|\mathcal{L}_{\text{CAN}}| = O((nk)^2)$. Thus the lemma follows. \square

Theorem 3.11. *For a given line segment \overline{pq} and a set P of n points in the Euclidean plane, we can solve the COFL problem optimally (Algorithm 2) in $O((nk)^2)$ time.*

Proof. From Lemma 3.10, we know that the cardinality of the set \mathcal{L}_{CAN} of all candidate radii is finite and is $O((nk)^2)$. Hence, we find $r_{\max} \in \mathcal{L}_{\text{CAN}}$ by repeatedly applying the selection algorithm for finding the median of \mathcal{L}_{CAN} and removing half of the elements in \mathcal{L}_{CAN} based on the output of the decision algorithm each time. This will take $O((nk)^2)$ time and then invoking the decision algorithm $O(\log(nk))$ times results in overall time of $O((nk)^2 + (n \log n + k) \log(nk)) = O((nk)^2)$. \square

3.3.2 Algorithm based on parametric search

A formally given FPTAS for the COFL problem is weakly polynomial because the number of calls to the decision algorithm is proportional to the number of bits of precision of accuracy for the desired approximation. Whereas the latter exact algorithm (in the previous subsection) runs in $O(n^4)$ time if $k = O(n)$. Here, we discuss an algorithm based on (slower version of) Megiddo's parametric search [53] whose running time is independent of any numerical precision parameter ($\frac{1}{\epsilon}$) and also becomes $O(n^2)$ when $k = O(n)$, running faster than the latter. As we know, in a solution based on parametric search, there is a test algorithm and a decision algorithm wherein

Algorithm 2: *Exhaustive_Search*(P, k, \overline{pq})

Input: A set $P = \{p_1, p_2, p_3, \dots, p_n\}$ of n points, a line segment \overline{pq} in the plane and an integer k .

Output: The radius r_{max} of k disks packed with centers lying on \overline{pq} .

Initialize $\mathcal{L}_{CAN} = \emptyset \cup \left\{ \frac{\|\overline{pq}\|}{2(k-1)} \right\}$

// \mathcal{L}_{CAN} is the set of candidate radii and *Case 1* contributes one r_{CAN}

for $j = 1, \dots, k$ **do**

for each $p_i \in P$ **do**

 compute r_{CAN} by evaluating the equation 3.1 (the equation 3.2 for mirror case) in *Case 2*.

 update $\mathcal{L}_{CAN} = \mathcal{L}_{CAN} \cup \{r_{CAN}\}$

for $j = 1, \dots, k$ **do**

for $j' = j, \dots, k$ **do**

for each pair $p_i, p_{i'} \in P$, where $i \neq i'$ **do**

 compute r_{CAN} by evaluating the equation 3.3 in *Case 3*.

 update $\mathcal{L}_{CAN} = \mathcal{L}_{CAN} \cup \{r_{CAN}\}$

while $|\mathcal{L}_{CAN}| \geq 2$ **do**

$L \leftarrow \text{median}(\mathcal{L}_{CAN})$ // invoke the linear-time median finding algorithm [18]

if *Greedy_LPacking*(P, k, L) **then**

 // Algorithm 1 returns YES

$\mathcal{L}_{CAN} \leftarrow \mathcal{L}_{CAN} \setminus \{x \in \mathcal{L}_{CAN} \mid x < L\}$

else

$\mathcal{L}_{CAN} \leftarrow \mathcal{L}_{CAN} \setminus \{x \in \mathcal{L}_{CAN} \mid x \geq L\}$

$r_{max} \leftarrow \text{element}(\mathcal{L}_{CAN})$

// *element*(\mathcal{L}_{CAN}) returns the last element of \mathcal{L}_{CAN} when $|\mathcal{L}_{CAN}| = 1$

return r_{max}

the test algorithm is typically a step-by-step simulation of the decision algorithm. We now will describe how to simulate the steps of Algorithm 1 at the unknown maximum $L^*(= r_{max})$.

Consider a point $p_i \in P$ which is having two center points $c_{i,1}$ and $c_{i,2}$ on the segment \overline{pq} (see Figure 3.4). Let the coordinates of these points be $p_i = (x(p_i), y(p_i))$, $c_{i,1} = (x_{i,1}, y_{i,1})$ and $c_{i,2} = (x_{i,2}, y_{i,2})$. Clearly, these points and L satisfy the equations:

$$L^2 = (x(p_i) - x_{i,1})^2 + (y(p_i) - y(q))^2 \quad (3.4)$$

$$L^2 = (x(p_i) - x_{i,2})^2 + (y(p_i) - y(q))^2 \quad (3.5)$$

where $y_{i,1} = y_{i,2} = y(p) = y(q)$ as both the points $c_{i,1}$ and $c_{i,2}$ are located on \overline{pq} . Since we know the coordinate values $y(q)$, $x(p_i)$ and $y(p_i)$, the values of $x_{i,1}$, $x_{i,2}$ are given as follows:

$$x_{i,1} = x(p_i) - (L^2 - (y(p_i) - y(q))^2)^{\frac{1}{2}} \text{ and}$$

$$x_{i,2} = x(p_i) - (L^2 - (y(p_i) - y(q))^2)^{\frac{1}{2}}$$

respectively, if

$$y(p_i) \leq y(q) + L \quad (3.6)$$

Now, consider the end points of the i th complemented interval as $[r_{i-1}l_i]$ (at line 5 of Algorithm 1). Let the coordinates $r_{i-1} = (x_{i-1,2}, y_{i-1,2})$, and $l_i = (x_{i,1}, y_{i,1})$, then

$$\begin{aligned} \gamma &= (||r_{i-1}l_i||)/2L = ((x_{i,1} - x_{i-1,2}))/2L, \\ \gamma &= (x(p_i) - (L^2 - (y(p_i) - y(q))^2)^{\frac{1}{2}} - \\ &\quad (x(p_{i-1}) - (L^2 - (y(p_{i-1}) - y(q))^2)^{\frac{1}{2}}))/2L. \end{aligned}$$

In the *for*-loop, at line 4 of Algorithm 1, we know the values of j and k . With these known values, we need to perform the comparison $j + \gamma + 1 - k \leq 0$. Note that this is a branching point depending on a comparison that involves a polynomial in L , $poly(L)$.

$$\begin{aligned} j + (x(p_i) - (L^2 - (y(p_i) - y(q))^2)^{\frac{1}{2}} - (x(p_{i-1}) - \\ (L^2 - (y(p_{i-1}) - y(q))^2)^{\frac{1}{2}}))/2L + 1 - k \leq 0, \\ 2(j + 1 - k)L + (x(p_i) - x(p_{i-1})) \leq \\ (L^2 - (y(p_i) - y(q))^2)^{\frac{1}{2}} - (L^2 - (y(p_{i-1}) - y(q))^2)^{\frac{1}{2}}. \end{aligned}$$

On simplification, the above inequality becomes

$$\begin{aligned} 2((j + 1 - k)^2 - 1)L^2 + 4(j + 1 - k)(x(p_i) - x(p_{i-1}))L + (x(p_i) - x(p_{i-1}))^2 \\ + 2\{(L^2 - (y(p_i) - y(q))^2)(L^2 - (y(p_{i-1}) - y(q))^2)\}^{\frac{1}{2}} \leq 0. \end{aligned}$$

that involves a degree-two polynomial, $poly(L)$: $AL^2 + BL + C + 2((L^2 - D)(L^2 - E))^{\frac{1}{2}} = 0$, where the coefficients A , B , C , D and E depend on the values known at the point of execution of the corresponding comparison step. To simulate comparison steps in the *for* loop at line 6, we compute all the roots of the associated polynomial $poly(L)$ and invoke Algorithm 1 with the value of L equal to each of these roots. This yields an interval between two roots or a root and 0 or $||pq||/(2(k - 1))$ that contains L^* . This enables us to determine the sign of $poly(L^*)$ and proceed with the generic execution of the next step of the algorithm. Essentially, each time Algorithm 1 returns YES for the guessed value of L , the region \mathcal{R} (which depends on the right end of the interval containing L^*) is shrinking, and hence the number of complemented intervals in I^c on which we have to run Algorithm 1 is reducing, until the shrunken \mathcal{R} corresponds to L^* . Finally, after completing the simulation of the *for* loop, if $j = k$, then we return L^* . To construct the set $D = \{d_1, d_2, \dots, d_k\}$, we run Algorithm 1 with the computed L^* one more time. Therefore, we have the following theorem.

Theorem 3.12. *We have an algorithm to solve the COFL problem in $O((n \log n + k)^2)$ using the parametric search technique.*

Proof. We know that Algorithm 1 runs in $O(n \log n + k)$ time. In the worst case, for each step of Algorithm 1, we obtain two different values of L and invoke Algorithm 1 on each of them as the candidate radius. Let L_1, L_2, \dots, L_t be those different values of L across the entire simulation, where $t = O(n \log n + k)$. Then, initially $r_{max} \in [0, \frac{\|pq\|}{2(k-1)}]$. After the entire simulation is completed, clearly $r_{max} = L^* = \max\{L_u \mid \text{DCOFL}(P, k, L_u) = \text{YES}, u = 1, 2, \dots, t\}$. Since the degree of the polynomial $poly(L)$ is at most two and the decision algorithm $\text{DCOFL}(P, k, L_u)$ is monotone for any $L_u \in \mathbb{R}^+ \cup \{0\}$, the entire setup fits in the framework of parametric search. Hence, the correctness of the algorithm follows, and the overall time for the simulation is $O((n \log n + k)^2)$. \square

3.3.3 Improved algorithm for $k = 2$

Here, we show that the decision problem $\text{DCOFL}(P, 2, L)$ can be solved in $O(\log n)$ parallel time using n processors. Let $Q(L, P)$ be the complement of the union of n open disk of radius L centered at the demand points in P . Let $S(\overline{pq}, L)$ be the intersection of $Q(L, P)$ and \overline{pq} , which is the collection I^c of $O(n)$ disjoint feasible intervals $[r_{i-1}l_i] \subset \overline{pq}$, $i = 1, 2, \dots, m$, where the coordinates $r_{i-1} = (x_{i-1,2}, y_{i-1,2})$, and $l_i = (x_{i,1}, y_{i,1})$ and $m = O(n)$. Let an infeasible interval $[x_{i,1}, x_{i,2}]$ be an interval on \overline{pq} , which is not feasible for centering a facility in that (excluding its endpoints). When we have the infeasible intervals computed implicitly, we also have computed the feasible intervals in I^c . A parallel algorithm (see Algorithm 3) for computing these intervals is as follows: (1) We assign one demand point from P to each of the n processors. (2) Let each processor compute the corresponding infeasible interval $[x_{i,1}, x_{i,2}] = d_i \cap \overline{pq}$, where d_i is the open disk of radius L centered at $p_i \in P$ for $i = 1, 2, \dots, n$. (3) Then, the merging of consecutive overlapping infeasible intervals into one bigger infeasible interval is performed by processors as follows: If an interval is not overlapping with its adjacent intervals, then this interval is maintained on the same processor and for each pair of consecutive overlapping intervals, the processor of the first interval merges them into one and the second processor sits idle. In this way, for a sequence of consecutive overlapping intervals, alternating processors will perform the merging. This process will be repeated until there are only isolated intervals. Since in every step, we merge a pair of consecutive overlapping intervals, there will be at most $\log n + 1$ steps in total, and each step will take $O(1)$ parallel time. Also, the serial time to construct $S(\overline{pq}, L)$ is only $O(n \log n)$ (see Algorithm 1).

Now, choosing the two farthest points on the intervals in $S(\overline{pq}, L)$ is easy, just pick the left endpoint of the leftmost interval and right endpoint of the rightmost interval in I^c , in $O(1)$ parallel time, and place the two facilities centered at these points.

Therefore, overall time of the parametric algorithm to solve COFL for $k = 2$ is $O(T_p \cdot n \cdot \log n + T_p \cdot T_s \cdot \log n) = O(\log n \cdot n \cdot \log n + \log n \cdot n \log n \cdot \log n) = O(n \log^3 n)$ time, where T_p

Algorithm 3: PARALLEL_DECISION($P, 2, L$)

Setup: Let $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ be the given n processors (in a shared memory multiprocessing system) ordered according to the demand points p_1, p_2, \dots, p_n such that the points are assigned as $P_1 \leftarrow p_1, P_2 \leftarrow p_2, \dots, P_n \leftarrow p_n$. Let \mathcal{B}_n be a bit string where each bit corresponds to one processor. If the bit value $\mathcal{B}_n[i]$ is 1, then the corresponding processor P_i will be active in the next step, and otherwise ($\mathcal{B}_n[i] = 0$) P_i will not perform any computation in the remaining steps of the algorithm.

for $i \leftarrow 1$ **to** n **do**

$\mathcal{B}_n[i] \leftarrow 1$
 // assume \mathcal{B}_n is stored in memory such that it can be accessed by every processor

end

Let the demand points be stored in the consecutive memory blocks B^1, B^2, \dots, B^n of same size, respectively.

Each processor p_i will compute an infeasible interval $[l_i, r_i]$ on \overline{pq} and stores in respective memory block B^i by replacing the demand point p_i .

Each processor P_{2i+1} will test whether its interval $[l_{2i+1}, r_{2i+1}]$ overlaps $[l_{2(i+1)}, r_{2(i+1)}]$.

if overlapping then

P_{2i+1} will access the bit $\mathcal{B}_n[2(i+1)]$ and set it to 0.
 Each processor P_{2i+1} will merge the intervals if $([l_{2i+1}, r_{2i+1}] \cap [l_{2(i+1)}, r_{2(i+1)}] \neq \emptyset) \wedge (\mathcal{B}_n[2(i+1)] = 0)$ and store the resulting interval in B^{2i+1} , where $i = 0, 1, \dots, \lfloor \frac{n-1}{2} \rfloor$.

end

$j \leftarrow 1$

while $j \leq \lceil \log n \rceil$ **do**

Each processor P_{2i+1} will test whether its interval $[l_{2i+1}, r_{2i+1}]$ overlaps $[l_{2(i+j)+1}, r_{2(i+j)+1}]$.
 Each processor P_{2i+1} will merge the intervals if $([l_{2i+1}, r_{2i+1}] \cap [l_{2(i+j)+1}, r_{2(i+j)+1}] \neq \emptyset) \wedge (\mathcal{B}_n[2(i+j)+1] = 0)$ and store the resulting interval in B^{2i+1} , where $i = 2t \leq \lfloor \frac{n-1}{2} \rfloor$ for $t = 0, j, 2j, 3j, \dots$
 $j \leftarrow j + 1$

end

Let $[l_l^*, r_l^*]$ and $[l_r^*, r_r^*]$ be the leftmost and rightmost infeasible intervals of the corresponding active processors P_l and P_r , which will perform the following steps.

if $(l_l^* \geq p) \wedge (r_r^* \leq q)$ **then** $l^* \leftarrow p, r^* \leftarrow q$.

if $(l_l^* < p) \wedge (r_r^* > q)$ **then** $l^* \leftarrow r_l^*, r^* \leftarrow l_r^*$.

if $(l_l^* \geq p) \wedge (r_r^* > q)$ **then** $l^* \leftarrow p, r^* \leftarrow l_r^*$.

if $(l_l^* < p) \wedge (r_r^* \leq q)$ **then** $l^* \leftarrow r_l^*, r^* \leftarrow q$.

if $\|l^* r^*\| \geq 2L$ **then**

return (YES, $\{d_1, d_2\}$)
 // where d_1 and d_2 with radius L are centered at l^* and r^* , respectively

end

else

return (NO, \emptyset)

end

denotes a parallel time and T_s denotes a serial time for solving $\text{DCOFL}(P, 2, L)$ with n processors. In essence, the problem solved by the parallel algorithm is the ordering of the endpoints of the intervals of $S(\overline{pq}, L)$ from left to right so that a pair of endpoints apart by at least $2L$ distance, if existing, can be found in $O(1)$ parallel time. Since the problem essentially involves sorting without knowing the optimal radius, using Cole's parametric search technique [16] (which trims a factor of $O(\log n)$) results in an improvement in the running time for $k = 2$, that is $O(n \cdot T_p + T_s(T_p + \log n)) = O(n \cdot \log n + n \log n(\log n + \log n)) = O(n \log^2 n)$.

Remark 3.13. The COFL under the Euclidean norm for $k = 2$ can be solved in $O(n \log^2 n)$ time using Cole's parametric search [16].

This is an improvement over the earlier FPTAS (proposed in subsection 3.2) as well as the two exact algorithms (proposed in subsection 3.3) for $k = 2$.

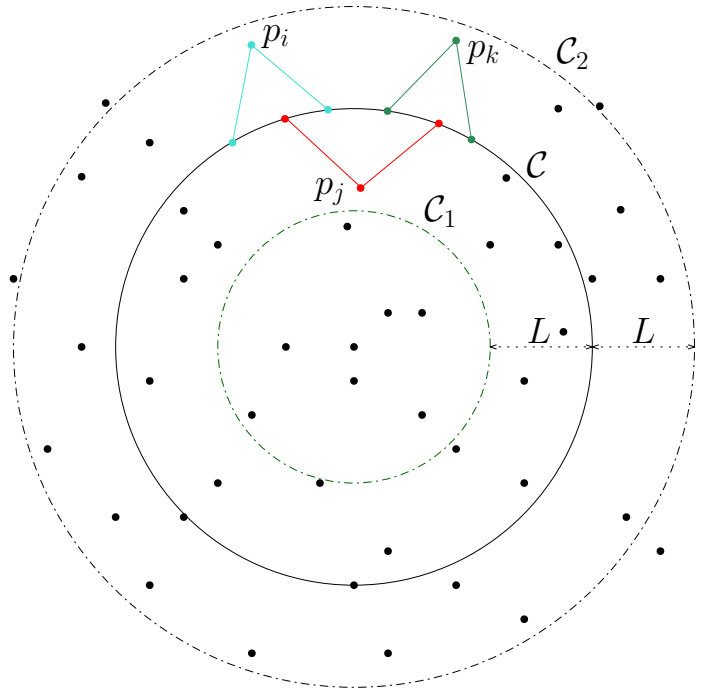
3.4 Circular COFL problem

Recall the definition of the CCOFL problem in which the centers of the disks are restricted to lie on the boundary arc $(\partial\mathcal{C})$ of a predetermined circle \mathcal{C} with radius r_c .

The decision version of this problem $\text{DCCOFL}(P, k, L)$ can be solved by using a similar method that was used to solve the $\text{DCOFL}(P, k, L)$ problem under the assumption that $r_c \gg L$ (r_c is much larger compared to L). When $r_c < L$, it is trivial that only one disk with radius L will be packed on $\partial\mathcal{C}$ as the center of \mathcal{C} will lie inside that packed disk. Now, to solve the decision version of CCOFL problem, we consider two circles \mathcal{C}_1 and \mathcal{C}_2 which are concentric with \mathcal{C} and whose radii are $r_c - L$ and $r_c + L$, where L is a real number (see Figure 3.10). We can observe that the points lying inside \mathcal{C}_1 and outside of \mathcal{C}_2 will not influence the packing of the disks. Similar to that of $\text{DCOFL}(P, k, L)$ problem here, we can obtain at least one center point and at most two center points on the boundary of \mathcal{C} which are at a distance of L from each point lying outside of \mathcal{C}_1 and inside of \mathcal{C}_2 (see p_i, p_j and p_k in Figure 3.10).

Let $c_{i,1}$ and $c_{i,2}$ be the center points corresponding to p_i , then none of the k disks in an optimal solution to $\text{DCCOFL}(P, k, L)$ will have their center points lying on the open arc interval $(c_{i,1}, c_{i,2})$ of the boundary of \mathcal{C} .

Now, let $(c_{j,1}, c_{j,2})$ and $(c_{k,1}, c_{k,2})$ be the center points corresponding to p_j and p_k respectively. In Figure 3.10, we can observe that the intervals $[c_{i,1}, c_{i,2}]$, $[c_{j,1}, c_{j,2}]$ and $[c_{k,1}, c_{k,2}]$ formed by $\widehat{c_{i,1}c_{i,2}}$, $\widehat{c_{j,1}c_{j,2}}$ and $\widehat{c_{k,1}c_{k,2}}$ are overlapping. Hence, none of the k disks in the optimal solution will have their centers lying on the interval $([c_{i,1}, c_{i,2}] \cup [c_{j,1}, c_{j,2}] \cup [c_{k,1}, c_{k,2}]) \setminus \{c_{i,1}, c_{k,2}\}$, excluding the end-points of the union of the two intervals.

FIGURE 3.10: \mathcal{C}_1 and \mathcal{C}_2 which are at distance L from \mathcal{C} .

Without loss of generality, let $\{p_1, p_2, \dots, p_m\}$ be the points of P lying strictly outside of \mathcal{C}_1 and inside of \mathcal{C}_2 , ordered along the boundary of \mathcal{C} in clockwise angular fashion, where $m \leq n$. We know that for every point p_i lying strictly outside of \mathcal{C}_1 and inside of \mathcal{C}_2 there will be two center-points on the boundary of \mathcal{C} which are at distance L , i.e., there is an interval $[l_i, r_i]$ for every point p_i , where $l_i = c_{i,1}$ and $r_i = c_{i,2}$. Merge all the overlapping intervals and then update the end-points of the new intervals on the boundary of \mathcal{C} . Let $I = \{[l_1, r_1], [l_2, r_2], \dots, [l_{m'}, r_{m'}]\}$ be the set of resulting pairwise disjoint intervals ordered clockwise, where $m' \leq m$. Consider the complement of I with respect to boundary of the circle \mathcal{C} , denoted as $I^c = \{[r_1, l_2], \dots, [r_{m'}, l_1]\}$.

The arc length of the complemented intervals in I^c can be calculated using the law of cosines formula as follows: The angle (θ) subtended by an arc at the center of \mathcal{C} is $\theta = \arccos(1 - \frac{d^2}{2r_c^2})$ where d is the Euclidean distance between the endpoints of the arc segment. Then, the length of the arc interval $[r_i l_{i+1}]$ is $|\widehat{r_i l_{i+1}}| = r_c \theta$, where $\theta = \arccos(1 - \frac{d^2}{2r_c^2})$ for $i = 1, 2, \dots, m'$.

Observation 3.14. Without loss of generality, we can assume that the first disk d_1 is centered at one of the end-points of the arc segment in I^c .

Since we don't know the position of the disks for given L in the optimal packing, we greedily pack disks by placing centers on $\partial\mathcal{C}$ with the first disk d_1 at every endpoint of the arc segments in I^c . As there are $O(n)$ end-points in I^c , we have the following theorem.

Lemma 3.15. *Given the set I^c of complemented intervals and an integer $k > 0$, Algorithm 1 solves the DCCOFL(P, k, L) problem in $O(n(n \log n + k))$ time.*

Proof. Follows from Observation 3.14. □

Theorem 3.16. *We can get a $(1 - \epsilon)$ -factor approximation algorithm with $\epsilon > 0$ (FPTAS) for the CCOFL problem, that runs in $O(n(n \log n + k) \log(\frac{\|pq\|}{2^{(k-1)\epsilon}}))$ time, by employing doubling search and bisection methods.*

3.5 Min-sum Obnoxious Facility Location (MOFL) problem

Recall that in the MOFL problem, we have a horizontal segment \overline{pq} in the plane; without loss of generality, we can assume that all the points in P are lying above the horizontal line through \overline{pq} . As in the previous section, we again compute the intervals $[l_i, r_i]$ on \overline{pq} for every point $p_i \in P$, but now defined by the *center-points* l_i and r_i on \overline{pq} , each at a distance λ from p_i . We call these intervals *mega-intervals* denoted by $I^{mega} = \{[l_1, r_1], [l_2, r_2], \dots, [l_n, r_n]\}$. We have already shown (in subsection 3.1, page no. 20) that I^{mega} can be computed in $O(n \log n)$ time. Also, observe that a disk of radius λ centered anywhere on \overline{pq} but only within $[l_i, r_i]$, covers the point p_i .

To solve the MOFL problem for $k = 1$ we use the approach of Katz et al., [43], but here the mega-intervals are defined differently as we are placing a disk instead of a rectangle in [43]. Then, similar to their approach, we define the elementary intervals on \overline{pq} , defined by every consecutive pair of endpoints of the intervals of I^{mega} starting with the left-end point p of the segment \overline{pq} . We call these elementary intervals *mini-intervals*, denoted by $I^{mini} = \{[\mu_1, \tau_1], [\mu_2, \tau_2], \dots, [\mu_{2n+1}, \tau_{2n+1}]\}$. Observe that there can be at most $2n + 1$ mini-intervals defined by n points in P and $\tau_j = \mu_{j+1}$ for $j = 1, 2, \dots, 2n$. We define the weight of each mini-interval $[\mu_j, \tau_j]$ to be $\kappa_j = \sum_{\{i \mid [\mu_j, \tau_j] \subseteq [l_i, r_i] \in I^{mega}\}} w_i$, where w_i is the weight of the point $p_i \in P$, i.e., the sum of weights of all the points whose corresponding mega-intervals contain $[\mu_j, \tau_j]$ entirely within them (see Figure 3.11 for an illustration of mega-interval, mini-interval, and their weights). Observe that an optimal disk d , i.e., the disk d of radius λ such that $\sum_{\{i \mid p_i \in d\}} w_i$ is minimized, can be centered anywhere in the mini-interval $[\mu_j, \tau_j]$ whose κ_j is minimal. As in [43], to efficiently find such a $[\mu_j, \tau_j]$ we construct a segment tree data structure on the mini-intervals $[\mu_j, \tau_j] \in I^{mini}$.

The construction of the segment tree and finding of minimum κ_j is briefly described as follows. We first construct a balanced binary tree T whose leaves correspond to the mini-intervals that are ordered from left to right. Since T is balanced and has n leaves, its depth must be $O(\log n)$. Further, we associate each node v of T with two attributes: (1) an interval which is the union of the mini-intervals of all the leaves of the subtree rooted at v , and (2) the weight which is equal to the sum of the minimum of the weights of its two child nodes and the weights of the mega-intervals that are stored in v . Initially, the weights of all the nodes of T , including the

leaves, are zero. The intervals of the leaves are their mini-intervals. A mega-interval will be stored at a node v if the interval of v is completely contained in it, and if so, then not at any of its descendant nodes. We now insert all the mega-intervals of I into T one by one, and during each insertion, we update the weight attributes of the nodes. The key feature of the segment tree is that each insertion of a mega-interval and the corresponding updates in T takes $O(\log n)$ time. At each insertion, the weight of the root of T is the smallest κ_j . The corresponding mini-interval $[\mu_j, \tau_j]$ can be found by traversing down the tree in the direction of the child with a smaller attribute weight, where ties can be broken arbitrarily. This traversal clearly takes $O(\log n)$ time. The mini-interval (not necessarily unique) corresponding to the weight of the root of the segment tree is the location for placing the disk d of radius λ such that $\sum_{\{i \mid p_i \in d\}} w_i$ is minimized. The construction of the segment tree takes $O(n \log n)$ time. Hence, we have the following theorem.

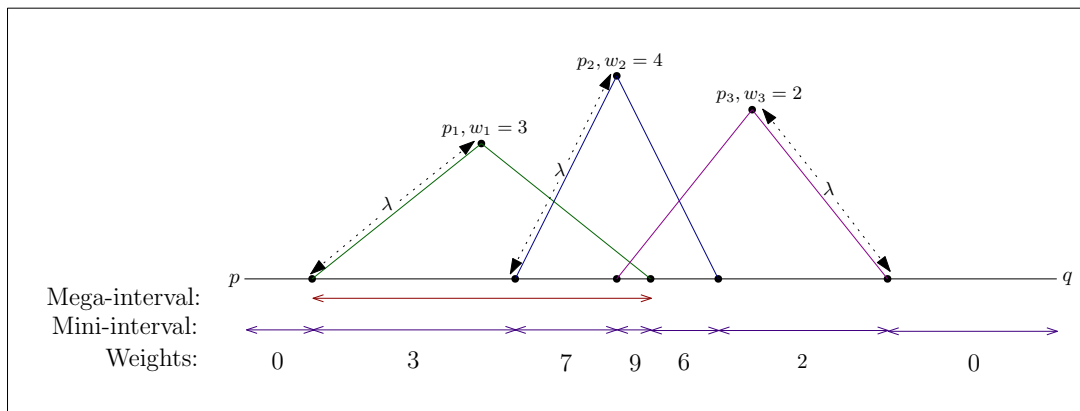


FIGURE 3.11: An illustration of splitting of mega-intervals into mini-intervals and their weight calculations.

Theorem 3.17. *The MOFL problem for $k = 1$ can be solved in $O(n \log n)$ time.*

Remark 3.18. A lower bound of $\Omega(n \log n)$ for the MOFL problem with $k = 1$ can be obtained using the same reduction as Katz et al., [43] did for their version of the obnoxious facility location problem.

3.5.1 Dynamic programming solution for any $k > 0$

Here, we first discuss a dynamic programming recurrence for computing the minimum weight of the points covered by k non-overlapping disks of radius λ centered on \overline{pq} . Then, we discuss how to actually reconstruct the solution, i.e., the mini-intervals on which the k disks of radius λ can be packed such that the sum of weights of the points covered by the union of these disks is minimal.

We now define the subproblem as follows: let $C(i, j, h)$ denote the minimum weight of the points covered by j non-overlapping disks of radius λ centered on the sub-segment $\overline{p\tau_i} = \bigcup_{s=1}^i [\mu_s, \tau_s]$, where the $j+1$ th disk was centered on the mini-interval $[\mu_t, \tau_t]$ such that $\|\mu_i\tau_{t-1}\| = h$. Clearly, $i \in \{1, 2, \dots, 2n+1\}$, $j \in \{1, 2, \dots, k\}$, and $h \in \{\|\mu_\alpha\tau_\beta\| \mid \alpha < \beta\}$.

To define the recurrence that relates between the subproblems (WLOG), we can assume that all the disks in an optimal solution are centered close to the left-end points of their respective mini-intervals ($\kappa_l \neq 0$ for every mini-interval $[\mu_l, \tau_l]$) and that $C(i, j, h) = \infty$ denotes the impossibility. Now, for computing $C(i, j, h)$, observe that the minimum-weighted packing of $j (\leq k)$ disks of radius λ either centers a disk on the mini-interval $\overline{\mu_i\tau_i}$ or not, but never centers a disk on $\overline{\mu_i\tau_i}$ if $h < 2\lambda$. Therefore, we have the recurrence below.

$$C(i, j, h) = \min \begin{cases} \kappa_i + C(i-1, j-1, \|\mu_{i-1}\tau_{i-1}\|) & \text{if } h \geq 2\lambda \\ C(i-1, j, h + \|\mu_{i-1}\tau_{i-1}\|) & \text{always} \end{cases}$$

The base cases are the following:

- If $i = 1$, $j = 1$, and $h \geq 2\lambda$, then $C(i, j, h) = \kappa_i$.
- If $i = 1$, $j = 1$, and $h < 2\lambda$, then $C(i, j, h) = \infty$.
- If $i > 1$, $C(i, 1, h)$ can be computed in $O(n \log n)$ time using Theorem 3.17.

The minimum weight of the points covered by the optimal packing can be obtained from $C(2n+1, k, 2\lambda+1)$. The number of possible values for h is $\binom{2n+1}{2}$. We can reconstruct the minimum-weighted packing (and their corresponding mini-intervals) by storing parent pointers between the entries in the three-dimensional array $C[2n+1, k, \binom{2n+1}{2}]$. From the above recurrence, it is clear that a subproblem depends on subproblems with strictly smaller i , and hence there is no cyclic dependency between the subproblems. We can fill the three-dimensional array by start filling its lower indexed entries first. Since computing the value of an entry requires the values of two lower indexed entries of the array, the work per subproblem is $O(1)$. Therefore, we have the following theorem.

Theorem 3.19. *The MOFL problem for any $k \geq 1$ can be solved in $O(n^3k)$ time using a dynamic programming approach.*

3.6 Conclusion

In this chapter, we gave an FPTAS for the COFL problem and presented two polynomial-time algorithms. The first algorithm is based on the binary search that runs in $O((nk)^2)$ time. The second algorithm is based on the parametric search that runs in $O((n \log n + k)^2)$ time. For $k = 2$ we gave $O(n \log^2 n)$ time algorithm. Additionally, we explored the circular variant of the problem and presented an FPTAS for it. Lastly, we presented $O(n^3k)$ time algorithm for the MOFL problem, which is based the dynamic programming.

The results in this chapter are published in [69], [70], and [71].

Chapter 4

Semi-Obnoxious Facility Location on a Line

In this chapter, we investigate many variations of the semi-obnoxious facility location (SOFL) problems defined based on whether the centers of the facilities are constrained to lie on a horizontal line, t distinct horizontal lines, and given points in convex position in the plane. Let d^0 denote the interior of any geometric object d , i.e., $d^0 = d \setminus \partial d$ where ∂d denotes the boundary of d . Firstly, we examine the variant that is restricted to a line, which is defined as follows:

Given a set \mathcal{B} of blue points and a set \mathcal{R} of red points, where each point $p_i \in \mathcal{B}$ has a weight $w_i > 0$ and each point $p_i \in \mathcal{R}$ has a weight $w_i < 0$, and let $|\mathcal{B} \cup \mathcal{R}| = n$. Assume these points lie above a given horizontal line ℓ . The objective is to pack k non-overlapping congruent disks d_1, d_2, \dots, d_k of minimum radius, centered on ℓ , such that the sum of the weights of the points covered by $\bigcup_{j=1}^k d_j$ is maximized, i.e., $\sum_{j=1}^k \sum_{i \in [n]: \exists p_i \in \mathcal{R}, p_i \in d_j^0} w_i + \sum_{j=1}^k \sum_{i \in [n]: \exists p_i \in \mathcal{B}, p_i \in d_j} w_i$ is maximized. We name this problem a Constrained Semi-Obnoxious Facility Location (CSOFL) problem on a Line.

We first show that the CSOFL can be solved optimally in $O(n^4 k^2)$ time. Subsequently, we improve the running time to $O(n^3 k \cdot \max(n, k))$. Furthermore, we addressed two special cases of the problem where points have only two types of weights, and we show that these two special cases can be solved in $O(n^3 k \cdot \max(\log n, k))$ time. For the first case, when $k = 1$, we also provide an algorithm that solves the problem in $O(n^3)$ time, and subsequently, we improve this result to $O(n^2 \log n)$. Furthermore, we consider a generalization of the weighted problem where we are given t horizontal lines instead of a single line ℓ . We give an $O(n^4 k^2 t^5)$ time algorithm for this problem. Finally, we consider a discrete variant where a set of s candidate sites (in convex position) for placing k facilities is pre-given ($k < s$). We propose an algorithm that runs in $O(ns(ns + s^4 k^2))$ time for this discrete variant.

These problems have a bi-chromatic objective, where we need to maximize the sum of weights covered by the disks, but these weights come from two opposing sets (i.e., positive weights and negative weights). The task is to locate centers for k disks on a given line such that the total weight covered by these disks is maximized, where there are infinitely many candidate centers for k disks. Here, the challenge lies in the fact that we need to pack k disks centered on the given line, covering as many positively weighted (blue) points as possible while simultaneously avoiding as many negatively weighted (red) points as possible.

4.1 Preliminaries

This section briefly introduces various notations and definitions that will be used in further sections.

Let \mathcal{L}_{CAN} denote the set of candidate radii and $r_{\text{CAN}} \in \mathcal{L}_{\text{CAN}}$ a candidate radius. The optimal radius is denoted as r_{opt} . Let $\text{dist}(u, v)$ denote the minimum Euclidean distance between two points u and v . Given a graph $G(V, E)$, the weight of an edge is denoted as $w(i, j)$ where $\overline{ij} \in E$. The path between any two vertices $v, u \in V$ is denoted as $\Pi(v, u)$.

Definition 4.1. Configurations: Arrangement of k -disks in any feasible solution to the CSOFL problem with different placements of red and blue points on the boundaries of the disks (critical regions). A configuration is said to be critical if it corresponds to some candidate radius $r_{\text{CAN}} \in \mathcal{L}_{\text{CAN}}$.

Definition 4.2. DAG: It stands for Directed Acyclic Graph. It is a directed graph that has no directed cycles. In other words, it is a graph consisting of a set of nodes connected by directed edges, where the edges have a specific direction. There is no way to start at any node and follow a sequence of edges that eventually loops back to that node.

Definition 4.3. The minimum weight k -link path: Given a complete weighted DAG $G(V, E)$, and two vertices s (source) and t (target), the minimum weight k -link path problem seeks to find a minimum weight path from s to t such that the path has exactly k edges (links) in it.

Definition 4.4. Concave Monge property: The weight function w for a given weighted, complete DAG $G(V, E)$ satisfies the concave Monge property if for all $i, j \in V$ we have the inequality $w(i, j) + w(i + 1, j + 1) \leq w(i, j + 1) + w(i + 1, j)$ satisfied, where $1 < i + 1 < j < n$.

The outline of the algorithm for the CSOFL problem is as follows:

1. First, all possible configurations of the k disks and red and blue points in any feasible solution to the CSOFL problem are identified. We show that a finite number of distinct configurations exist, and there are specifically $O(1)$ distinct critical-configuration types.

2. The next step entails computing all possible candidate radii, \mathcal{L}_{CAN} , where we have one r_{CAN} corresponding to each of the configurations identified in the previous step.
3. After obtaining a candidate radius r_{CAN} , the given instance of the CSOFL problem will be transformed into an instance of the problem of computing a minimum weight k -link path problem on a complete weighted DAG G .
4. The semi-obnoxious facilities (disks) should then be positioned (i.e., the centers of these disks are to be positioned) at the points on ℓ corresponding to vertices of the aforementioned minimum weight k -link path $\Pi_k^*(s, t)$ in G . The total weight of the points covered by these facilities can be computed using the $\Pi_k^*(s, t)$ weight.
5. To determine the set of all radii $\mathcal{L}_{\text{CAN}} = \{\lambda_1, \lambda_2, \dots\}$ for which the total weight of the covered points is the largest, the above process must be repeated for every candidate radius r_{CAN} .
6. Finally, the locations of the k semi-obnoxious facilities placed with the smallest $\lambda \in \mathcal{L}_{\text{CAN}}$ and covering the points with the largest total weight is returned as the output.

4.2 Computing the candidate radii

In this section, we find all the candidate radii by considering all configurations involving disks as well as blue and red points.

Configuration-0: Suppose that all the red points lie closer to ℓ than the blue points and have significantly more negative weights than the blue points (see Figure 4.1). In this scenario, covering any blue points would also cover some red points since the disks must be centered on ℓ . This, in turn, would result in a negative total weight. As a result, we can opt to keep zero-radius disks that do not cover any points rather than covering any of the blue points. This way, the maximum weight will be zero. It also implies the following observation.

Observation 4.5. An optimal (feasible) solution always exists for any given problem instance.

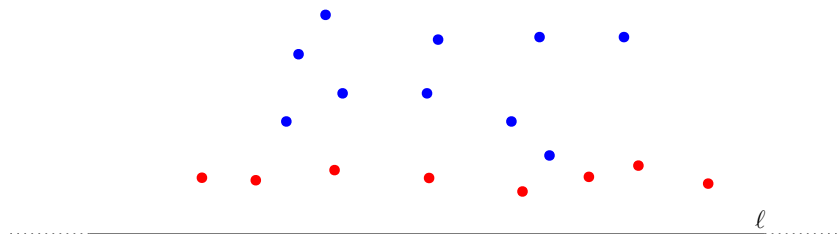


FIGURE 4.1: Configuration-0.

Configuration-1: We consider a specific configuration in which the radius of the disks in the optimal solution is determined by only one blue point. As shown in Figure 4.2, we can observe that the disks d_i and d_{i+1} , which have one blue point on each of their boundaries, will have a smaller (optimal) radius compared to the dotted disk d'_i , which also covers the same blue points. This is because our problem is to find the minimum radius disks that cover the maximum weight. Here, we consider at least one of the blue points lying on

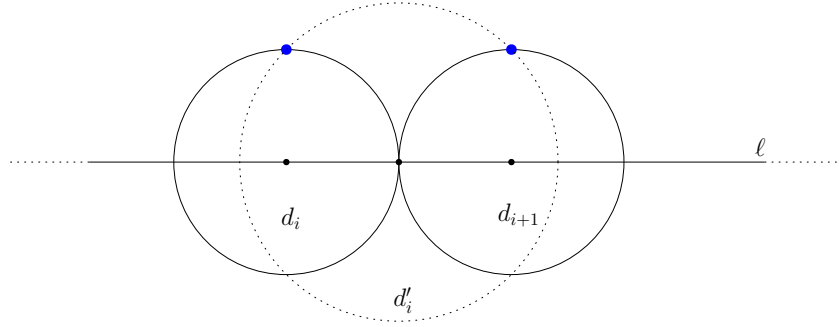


FIGURE 4.2: Configuration-1.

the boundary of at least one of the k disks, which determines the radius of the disks in the optimal solution for a radius greater than zero. Observe that the radius of the disk is the y -coordinate value of the point that lies on the disk boundary. Hence, we add $O(n)$ candidate radii to \mathcal{L}_{CAN} and the radii are $r_{\text{CAN}} = y_{p_i}$, where y_{p_i} denotes the y -coordinate of the point p_i for each $p_i \in \mathcal{B}$, $i = 1, 2, \dots, n$.

Configuration-2: In this scenario, we consider the case where the optimal solution is determined by two points on the boundary of at least one of the k disks, which can either be two blue points or one blue and one red point. We notice that no two red points on any disk's boundary will determine the disk's radius, as we can further reduce the disk's radius until its boundary touches at least one of the blue points. To calculate the candidate radii for the disks, we proceed as follows:

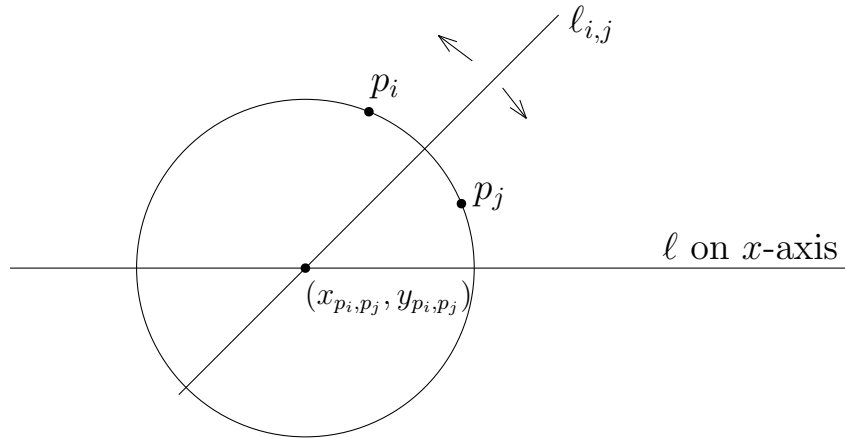
Consider a point $p_i \in \mathcal{B}$ and a point $p_j \in \mathcal{R}$. If they determine the minimum radius of the disks in a solution to CSOFL problem, then the candidate radius r_{CAN} can be computed by drawing a bisector line $\ell_{i,j}$ until $\ell_{i,j}$ cuts across ℓ where p_i is in the counter-clockwise direction from $\ell_{i,j}$ and p_j in the clockwise direction from $\ell_{i,j}$ (see Figure 4.3).

Let $(x_{p_i,p_j}, y_{p_i,p_j})$ be the center of the disk, (x_{p_i}, y_{p_i}) and (x_{p_j}, y_{p_j}) are the coordinates of the points p_i and p_j respectively. Then, we have

$$(x_{p_i} - x_{p_i,p_j})^2 + (y_{p_i} - y_{p_i,p_j})^2 = (x_{p_j} - x_{p_i,p_j})^2 + (y_{p_j} - y_{p_i,p_j})^2$$

After simplification, we have $r_{\text{CAN}} = \sqrt{(x_{p_i} - x_{p_i,p_j})^2 + y_{p_i}^2}$ for the two cases:

- $p_i \in \mathcal{B}$ and $p_j \in \mathcal{R}$.

FIGURE 4.3: Illustration of calculating r_{CAN} .

– $p_i, p_j \in \mathcal{B}$.

where $x_{p_i, p_j} = \frac{(y_{p_j} - y_{p_i})(y_{p_j} + y_{p_i})}{2(x_{p_j} - x_{p_i})} + \frac{(x_{p_j} + x_{p_i})}{2}$, $y_{p_i, p_j} = 0$. Here, as we consider a candidate radius for every pair of points except the red-red pair, we add $O(n^2)$ candidate radii to \mathcal{L}_{CAN} for this critical-configuration type.

Observation 4.6. There are only $O(1)$ critical configuration types.

Proof. In any of the optimal placements of the disks, either one blue point or a pair of blue points, or a pair of red and blue points will determine the disk's radius. Even though there may be more than two points lying on the boundary of the disks in any optimal packing, only two points among them will determine the radius of the disk since there exists a unique disk that passes through two points and is centered on ℓ . Hence, any configurations of points lying on the boundary of the disk can be transformed into any of the above-mentioned configurations by perturbing the center or reducing the radius of the disks. Therefore, we have a constant number of critical configuration types, namely, four types, including the configuration-0. \square

Lemma 4.7. $|\mathcal{L}_{\text{CAN}}| = O(n^2)$.

Proof. It follows from Observation 4.6 since a constant number of critical configuration types (viz. no points, one blue point, a pair of blue points, and a pair of blue and red points) contribute to the candidate radii. In any of the configurations, at most, two points will determine the radius of the disks. Hence we have $O(n^2)$ candidate radii corresponding to that configuration. Thus, the lemma follows. \square

4.3 Transformation to the minimum weight k -link path problem

In this section, we demonstrate that the CSOFL problem can be reduced to the problem of computing a minimum weight k -link path between a pair of vertices in a weighted DAG $G(V, E)$. Each edge $\overline{ij} \in E$ in G is assigned a weight $w : (i, j) \rightarrow \mathbb{R}$ that is either a positive or negative real number $w(i, j) \in \mathbb{R}$.

The minimum weight k -link path $\Pi(s, t)$ is a path from the source s to the target vertex t , consisting of exactly k edges, and has the minimum total weight among all k -link paths between s and t , where the weight of a k -link path is the sum of weights of the edges in the path, i.e.,

$$w(\Pi(s \rightarrow i_1 \rightarrow i_2 \rightarrow \cdots \rightarrow i_{k-1} \rightarrow t)) = \sum_{j=1}^{k-2} w(i_j, i_{j+1}) + w(s, i_1) + w(i_{k-1}, t)$$

Let $\lambda = r_{\text{CAN}}$. Next, we transform an instance of the CSOFL problem to an instance of k -link path problem on a DAG $G(V, E)$ as follows:

Let us call the maximal interval $f_i^+ = [l_i, r_i]$ on ℓ as the influence interval (within which a facility or a disk with radius r_{CAN} centered will influence or cover the point p_i) for the point $p_i \in \mathcal{B}$ if the distance between any point on $[l_i, r_i]$ and p_i is at most r_{CAN} . Similarly, $f_i^- = [l_i, r_i]$ is the influence interval on ℓ for $p_i \in \mathcal{R}$.

Let the set of all influence intervals be $F = \{f_i^+ \mid i \in [n], p_i \in \mathcal{B}\} \cup \{f_i^- \mid i \in [n], p_i \in \mathcal{R}\}$. Let the vertex set $V = \{l_1, r_1, l_2, r_2, \dots, l_n, r_n\}$ be the end points of the intervals in F . For each $l_i, i \in [n]$, we also add $2(k-1)$ extra vertices corresponding to points on ℓ at distance $l_i + 2\lambda, l_i + 4\lambda, \dots, l_i + 2(k-1)\lambda, l_i - 2\lambda, l_i - 4\lambda, \dots, l_i - 2(k-1)\lambda$, to V . Similarly, we add $2(k-1)$ vertices for every $r_i, i \in [n]$, placed at points at distance $r_i - 2\lambda, r_i - 4\lambda, \dots, r_i - 2(k-1)\lambda, r_i + 2\lambda, r_i + 4\lambda, \dots, r_i + 2(k-1)\lambda$. The addition of these extra $2(k-1)$ points on the sides of both endpoints of each influence interval is because it is possible to have disks centered at points on ℓ other than the endpoints of influence intervals in an optimal solution (see Figure 4.4 for an illustration). However, at least one disk must be centered at an endpoint in any optimal solution. In Figure 4.4, we can observe that the disks d_{i-1} and d_{i+1} are not centered at any endpoint of the intervals in F since none of the points in $\mathcal{B} \cup \mathcal{R}$ lie on their boundary.

Without loss of generality, the vertices may be relabeled as $V = \{v_1, v_2, \dots, v_m\}$ based on the increasing order of x -coordinates of all l_i and r_i for $i \in [n]$, and the extra added points, where $m = O(kn)$. Furthermore, we can update V so that all the corresponding points in V have distinct x -coordinates. Let s and t be the points placed on ℓ at a distance of $2k\lambda$ from the left endpoint of the leftmost interval l_1 and from the right endpoint of the rightmost interval r_L , respectively, where $[l_L, r_L]$ denotes the rightmost influence interval (see Figure 4.5). Note that s lies on the left of all the points in V , and t lies on the right of all the points in V .

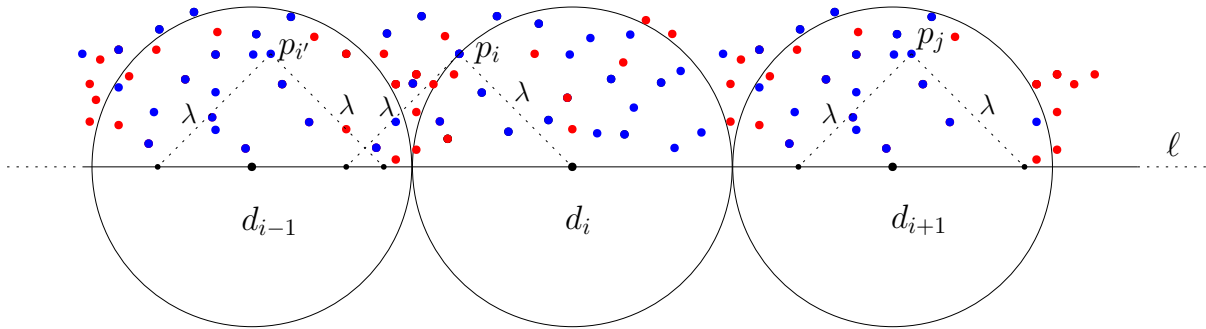


FIGURE 4.4: An optimal packing of 3 disks for a candidate radius λ , d_i is centered at an endpoint of the influence interval due to p_i .

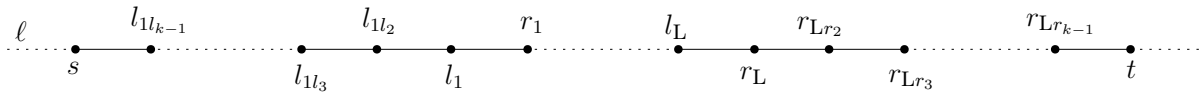


FIGURE 4.5: Adding of extra-points including s and t .

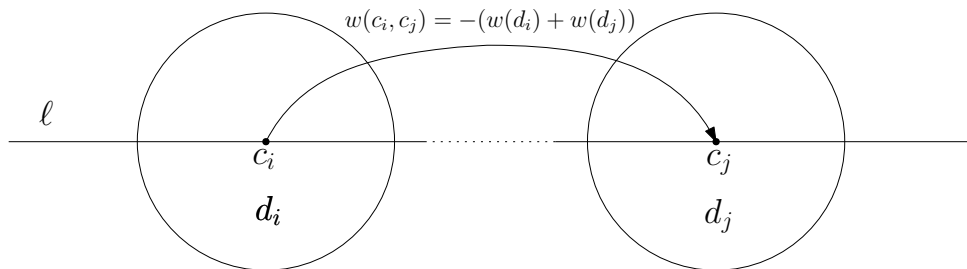


FIGURE 4.6: Calculation of the weight of an edge.

Let $w(d_i)$ denote the total weight of the points covered by the disk with radius λ centered at $c_i \in V$. We calculate $w(d_i)$ for the disks centered at each point $c_i \in V$.

Now, the weight of an edge $\overline{ij} \in E$ is calculated as follows:

- $w(i, j) = +\infty$ if the $\text{dist}(i, j) < 2\lambda$.
- $w(i, j) = -(w(d_i) + w(d_j))$ if $\text{dist}(i, j) \geq 2\lambda$ for all $i, j \in V$, i.e., we add a directed edge between every pair of vertices $i, j \in V$ ($i < j$), if the distance between them is at least 2λ and we add the corresponding weights (see Figure 4.6) and negate it.
- Clearly observe that $w(d_s)$ and $w(d_t)$ is zero. It is also zero for the disks centered at first (at most) $k - 1$ and last (at most) $k - 1$ points (since they are the points on ℓ which are separated by a distance of at least 2λ on the left of l_1 and the right of r_L), respectively for every endpoint of the influence interval.

Without loss of generality, let $G'(V', E')$ be the graph obtained by the above transformation. There will be $O(nk)$ vertices in V' , and a directed edge from i to j for all $i, j \in V'$ such that $i < j$. Then, G' is a complete DAG with $|V'| = O(nk)$ vertices and $|E'| = O(n^2k^2)$ edges, and every edge $(i, j) \in E'$ is assigned a weight as discussed above. Hence, we have the following lemma.

Lemma 4.8. *The graph G' can be constructed in $O(n^2k^2)$ time.*

Proof. We start by considering the way we constructed G' . Every demand point $p_i \in \mathcal{B} \cup \mathcal{R}$ can contribute at most two endpoints of an interval on ℓ at a distance of 2λ from each other. If we center a disk on that interval, the demand points will either lie on the boundary or the interior of the disk. Next, we add $2(k-1)$ points on both sides of each endpoint on ℓ with a separation distance of 2λ between any two consecutive of them. Thus, we have a total of $O(nk)$ points on ℓ , which includes s and t and are added to V' . Now, from every point in V' , we add a weighted directed edge to all the points of V' that lie on the right of that point on ℓ . This will result in a total of $O(n^2k^2)$ edges, where each edge is assigned a corresponding weight, as discussed earlier. Therefore, the resulting graph $G'(V', E')$ has $O(nk)$ vertices, $O(n^2k^2)$ edges, and can be constructed in $O(n^2k^2)$ time. \square

The edge weights of G' will satisfy the concave Monge property for any four vertices $i, i+1, j, j+1$ of G' such that $i < i+1 < j < j+1$, the weights of the directed edges from i to j and from $i+1$ to $j+1$ are not greater than the weights of the directed edges from i to $j+1$ and from $i+1$ to j .

Observation 4.9. The edge weights of G' satisfy the concave Monge property.

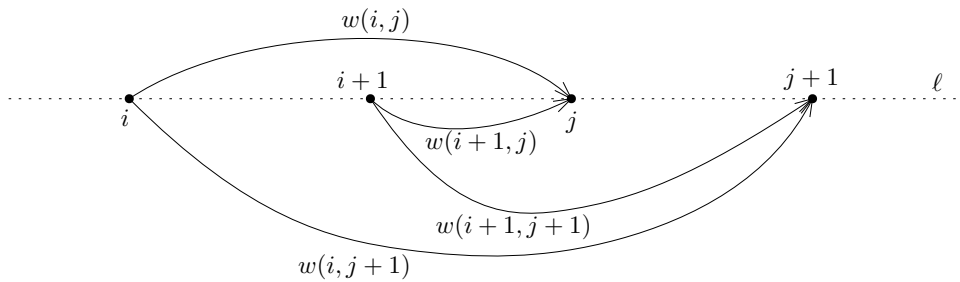


FIGURE 4.7: Any four vertices of G' that satisfy the concave Monge property.

Proof. Recall the definition of the concave Monge property, i.e., $w(i, j) + w(i+1, j+1) \leq w(i, j+1) + w(i+1, j)$ (see Figure 4.7). The weights assigned to the edges of G' are assigned based on the following two rules:

1. $w(i, j) = +\infty$ if the $\text{dist}(i, j) < 2\lambda$.
2. $w(i, j) = -(w(d_i) + w(d_j))$ if $\text{dist}(i, j) \geq 2\lambda$ for all $i, j \in V$

Suppose we select any four vertices with their index labels satisfying $i < i+1 < j < j+1$ in G' such that the distance between them is at least 2λ . Then, according to rule 2, the corresponding weights assigned to the edges satisfy $w(i, j) + w(i+1, j+1) = w(i, j+1) + w(i+1, j)$. Now, consider another set of four vertices $i < i+1 < j < j+1$ in G' such that the distance between the two closest points among them is less than 2λ , i.e., $\text{dist}(i+1, j) < 2\lambda$. According to rule 1,

the corresponding weight assigned to the edges between these vertices is $+\infty$. Then, we have $w(i, j) + w(i + 1, j + 1) < w(i, j + 1) + w(i + 1, j)$, which satisfies the concave Monge property. Finally, suppose all four selected vertices $i < i + 1 < j < j + 1$ in G' have a distance between any two consecutive of them less than 2λ . In this case, according to rule 1, the weights assigned to the corresponding edges satisfy $w(i, j) + w(i + 1, j + 1) = w(i, j + 1) + w(i + 1, j)$.

Therefore, we have shown that if we select any four vertices $i < i + 1 < j < j + 1$ in G' , the weights of all these edges satisfy the concave Monge property. Thus, the observation follows. \square

Since the constructed graph G' is a weighted complete DAG and its edge weights satisfy the concave Monge property, we have the following theorem for finding the minimum weight $(k+1)$ -link path between any pair of vertices of G' .

Theorem 4.10. [4] *The minimum weight $(k+1)$ -link path $\Pi_{(k+1)}^*(s, t)$ in G' can be computed in $O(nk\sqrt{k \log(nk)})$ time.*

Theorem 4.11. *We can solve the CSOFL problem in polynomial time.*

Proof. It follows from Lemma 4.7, Lemma 4.8 and Theorem 4.10. The running time of the algorithm is $n^2 \cdot (O(n^2k^2) + O(nk\sqrt{k \log(nk)})) = O(n^4k^2)$. The algorithm will return the minimum $r_{\text{CAN}} = r_{\text{opt}}$, corresponding to which the computed $(k+1)$ -link path between s and t has the minimum total weight $w(\Pi_{(k+1)}^*(s, t))$. The disks with radius r_{opt} can be centered at k internal vertices of the $(k+1)$ -link path (excluding the terminal vertices s and t). The total weight is $\sum_{i \in \text{Sol}} w(d_i) = -w(\Pi_{(k+1)}^*(s, t))/2$, where d_i is a disk centered at i having radius r_{opt} and $\text{Sol} = V(\Pi_{(k+1)}^*(s, t)) \setminus \{s, t\}$ is the set of vertices (the corresponding points on ℓ) of $(k+1)$ -link path except s and t . \square

Improvement: Recall that the points corresponding to the vertices in V' are labeled as $1, 2, \dots, m$, where $m = O(nk)$. Here, we show that we can improve the runtime of Theorem 4.11 (by almost linear factor) by not explicitly constructing the complete graph G' . As we have seen in the proof of Lemma 4.8, this construction requires $O(n^2k^2)$ time for each of $O(n^2)$ candidate radius. However, for every candidate radius r_{CAN} , we need to precompute the two arrays $w[]$ and $p[]$, each of size $O(nk)$. Here, $w[i]$ stores the sum of weights of the demand points covered by the disk of radius r_{CAN} centered at a point labeled $i \in V'$ on ℓ , for each $i \in [m]$. The element $p[i]$ stores the index $i' \in V'$ of the rightmost point at a distance of at least 2λ from i such that $i' < i$. We now give a dynamic program algorithm to compute the maximum weight of a $(k+1)$ -link path from point 1 to point m (here, the points labeled 1 and m are the vertices s and t respectively, in G'). For a pair of points i, j ($i < j$) on ℓ ; we redefine the weight of the edge (i, j) to be equal to $w(j)$ as we need not negate the sum of weights and construct the whole directed graph.

We define the subproblem $\phi(i, j)$ as the problem of finding the maximum weight j -link path in the subgraph G'_i induced by the vertices $1, 2, \dots, i$. That is, $\phi(i, j) = \max_{j'} \{w(\Pi_j(1, i'))\}$, where $(j+1) \leq i' \leq i$. Then we have the following recurrence:

$$\phi(i, j) = \max\{\phi(i-1, j), \phi(p[i], j-1) + w[i]\} \quad (4.1)$$

As $i = 1, 2, \dots, O(nk)$, and $j = 1, 2, \dots, k+1$, there are $nk \cdot k = O(nk^2)$ entries in the DP table $\phi(i, j)$, each requiring $O(1)$ time to compute.

Hence, for a given r_{CAN} , the bottom-up implementation of the above dynamic programming algorithm takes time $O(nk^2)$ provided that we have the entries $w[i]$ and $p[i]$ precomputed for each $i \in [m]$.

Theorem 4.12. *The above-improved algorithm for the CSOFL problem has the time complexity of $O(n^3k \cdot \max(n, k))$.*

Proof. The proof is as follows:

- There are $O(n^2)$ candidate radii in \mathcal{L}_{CAN} .
- For each candidate radius $\lambda \in \mathcal{L}_{\text{CAN}}$, we find $O(nk)$ points on ℓ as discussed above.
- To compute the weight $w[i]$ of each point $i \in V'$ on ℓ , we answer a circular range reporting query [13], which takes $O(\log n + \kappa)$ time for a query circle centered at each of $O(nk)$ points on ℓ , where κ is the number of points reported. It has a preprocessing time of $O(n \log n)$ and requires $O(n)$ space.
- For each $\lambda \in \mathcal{L}_{\text{CAN}}$, the above dynamic programming algorithm will take $O(nk^2)$ time. However, the bottleneck in computing the optimal value $\phi(m, k+1)$ is in computing the arrays $w[]$ for each of $O(n^2)$ candidate radius. The total time for computing this array is $n \log n + \sum_{j=1}^{|\mathcal{L}_{\text{CAN}}|} \sum_{i=1}^m (\log n + \kappa_i)$, where κ_i is the number of points reported by the query algorithm for a given query disk centered at $i \in [m]$, where $m = O(nk)$. We see that $\sum_{j=1}^{|\mathcal{L}_{\text{CAN}}|} \sum_{i=1}^m (\log n + \kappa_i) = n^3k \log n + \sum_{j=1}^{|\mathcal{L}_{\text{CAN}}|} n\chi_j$, where $\chi_j = \max\{\chi_1, \chi_2, \dots, \chi_{O(n^2)}\}$, and χ_j is the ply¹ of the pointset $\mathcal{B} \cup \mathcal{R}$ with respect to the set of all disks $i \in [m]$ for a candidate radius $\lambda_j \in \mathcal{L}_{\text{CAN}}$. Observe that the ply of $\mathcal{B} \cup \mathcal{R}$ for a given set of $O(nk)$ disks is $O(n)$ only since a point from $\mathcal{B} \cup \mathcal{R}$ lying in a disk centered at an endpoint i of influence interval can not be contained inside the $2(k-1)$ disks centered at distances

¹The ply of a set P of points with respect to a set D of disks d_1, d_2, \dots , is the largest number of those disks in $\bigcup_{i=1,2,\dots} d_i$ whose intersection contains a point $p \in P$.

$l_i + 2\lambda, l_i + 4\lambda, \dots, l_i + 2(k-1)\lambda, l_i - 2\lambda, l_i - 4\lambda, \dots, l_i - 2(k-1)\lambda$. Therefore, $\sum_{j=1}^{|\mathcal{L}_{\text{CAN}}|} \left(\sum_{i=1}^m \kappa_i \right) \leq \sum_{j=1}^{|\mathcal{L}_{\text{CAN}}|} (nk\chi) = O(n^4k)$ as $\chi = O(n)$.

- Hence the total running time is $n^2 \cdot (O(nk^2) + O(nk \log n) + O(n^2k)) = O(n^3k \cdot \max(n, k))$.

Now, we prove the correctness of the dynamic programming recurrence relation 4.1 by inducting on the number of disks placed.

Correctness: Fix a radius $\lambda \in \mathcal{L}_{\text{CAN}}$. By induction on $i + j$, we can prove the recurrence relation 4.1 is correct, as follows. For the base case $j = 1, i \geq 2$, we have $\phi(i, 1) = w(\Pi_1(1, i)) = \max_{2 \leq q \leq i} (w[q])$, which is the maximum weight of a 1-link path originating at vertex 1 in the subgraph G'_i induced by the vertices $1, 2, \dots, i$. This is the optimal solution for the subproblem $\phi(i, 1)$. For the base case $j \geq 2, i = 2$, we have that $\phi(i, j) = \phi(i, 2)$ as the weight contributed by the remaining $j - 2$ disks centered on ℓ is zero.

Let $p[s] = \max\{q \mid (\text{dist}(s, q) \geq 2\lambda, 2 \leq q < s)\}$ for $2 \leq s \leq i$. Assume that the recurrence relation holds for all subproblems $\phi(i', j')$, where $(i' + j') < (i + j)$. Consider the subproblem $\phi(i, j)$, and for solving this subproblem, we consider two cases for the vertex i : either $\Pi_j(1, i)$ uses vertex i or it doesn't.

Case 1: $\Pi_j(1, i)$ does not use vertex i . In this case, $\Pi_j(1, i)$ is also an optimal j -link path in G'_{i-1} by induction hypothesis. Therefore, the optimal solution for the subproblem $\phi(i, j)$ is the same as for the subproblem $\phi(i - 1, j)$.

Case 2: $\Pi_j(1, i)$ uses vertex i . Let $i' = p[i]$, which is the predecessor of i on $\Pi_j(1, i)$. Then, $\Pi_j(1, i)$ can be decomposed into two parts: an optimal $(j - 1)$ -link path in $G'_{i'}$ (by induction hypothesis), denoted by $\Pi_{j-1}(1, i')$, and the edge (i', i) with weight $w[i]$. Since $\Pi_j(1, i)$ is an optimal path ending at i in the subgraph G'_i , the weight of $\Pi_j(1, i)$ is equal to the sum of the weights of $\Pi_{j-1}(1, i')$ and (i', i) , i.e., $w(\Pi_j(1, i)) = w(\Pi_{j-1}(1, i')) + w[i]$.

Therefore, the optimal solution for the subproblem $\phi(i, j)$ is the maximum weight of all j -link paths in G'_i . This is achieved by either taking the optimal solution for the subproblem $\phi(i - 1, j)$ or by taking the optimal solution for the subproblem $\phi(i', j - 1)$ and adding the weight of edge (i', i) , i.e., $\phi(i, j) = \max(\phi(i - 1, j), \phi(p[i], j - 1) + w[i])$.

By using the recurrence relation for $\phi(i - 1, j)$ and $\phi(p[i], j - 1)$, which can be computed by solving the subproblems $\phi(i - 1, j - 1)$ and $\phi(p[i], j - 1)$. Therefore, we can use the recurrence relation to obtain the optimal solution for $\phi(i, j)$.

By the principle of mathematical induction, the recurrence relation holds for all subproblems $\phi(i, j)$, where $1 \leq j \leq k$. Given that we have precomputed all the values in the arrays $w[]$ and

$p[]$, it takes constant time to compute the optimal solution to each subproblem by combining optimal solutions to smaller subproblems. Further, we have $O(nk^2)$ distinct subproblems in total for the recurrence. Hence, the overall time complexity of the algorithm is $O(nk^2)$. \square

4.4 Special cases of CSOFL

In this section, we consider the following two special cases of the CSOFL problem with some specific application.

Problem 4.13. ALLBLUE-MINRED: The problem aims to cover all blue points while covering the minimum number of red points. To solve this problem, we modify the weights of the demand points as follows: for every point $p_i \in \mathcal{R}$, let $w_i = \delta$ and for every point $p_i \in \mathcal{B}$, the weight $w_i > -|\mathcal{R}|\delta$, where $\delta \in \mathbb{R}$ is an arbitrary real value and $\delta < 0$.

This problem has some specific applications in defense, as will be discussed: assuming a scenario where there are two groups of points along a horizontal line, one represented by blue points (enemy forces) and the other by red points (civilians), the goal is to determine the center locations and blast radius required for a fixed number of explosives to target all enemy forces while isolating the civilians as much as possible. Alternatively, suppose the scenario is such that the red points represent enemy forces, and the objective is to establish wireless communication among our own forces (represented by blue points). In that case, the goal is to place k base stations to cover all blue forces while minimizing the transmissions intercepted by the red forces (enemy forces).

Problem 4.14. MAXBLUE-NORED: In this problem, we need to cover the maximum number of blue points, and at the same time, none of the red points need to be covered. To solve this problem, we modify the weights of the demand points as follows: for every point $p_i \in \mathcal{B}$, let $w_i = \delta$, and for every point $p_i \in \mathcal{R}$, the weight $w_i < -|\mathcal{B}|\delta$, where $\delta \in \mathbb{R}$ is an arbitrary real value and $\delta > 0$.

This problem has the following specific applications: place a set of k sensors on a horizontal line to cover as many blue points as possible while avoiding red ones. This scenario can arise, for example, in battlefield surveillance, where the red points represent friendly forces, and the blue points represent enemy forces. The goal is to deploy sensors to monitor the enemy forces while avoiding the friendly forces. Similarly, the problem can arise in wildlife conservation, where the blue points represent areas of high animal activity, and the red points represent protected or private residential areas. The goal is to deploy sensors to monitor animal activity while avoiding private or protected areas.

Claim 4.15. The algorithm of Theorem 4.11 will eventually find an optimal solution (i.e., selects at most k facility locations on ℓ to cover all the blue points) for the ALLBLUE-MINRED problem.

Proof. Consider an instance of CSOFL with $w_i = \delta$ for every $p_i \in \mathcal{R}$ and the weight $w_i > -|\mathcal{R}|\delta$ for every $p_i \in \mathcal{B}$, where $\delta \in \mathbb{R}$ is an arbitrary negative real value.

Feasibility: The weight assignment of $w_i (> -|\mathcal{R}|\delta)$ guarantees that all blue points will be covered. In the worst-case scenario, a single disk can cover all points, both blue and red points, whose total weight is positive due to the weight assignment. The remaining $k - 1$ disks can be centered on ℓ to cover none of the points. This ensures that a feasible solution exists for the ALLBLUE-MINRED problem.

Optimality: Suppose there is a feasible solution for the CSOFL problem that places at most k facility centers on ℓ . Let these facilities cover all points in \mathcal{B} and some points in \mathcal{R} , with total weight equal to ρ . Now observe that it is impossible to improve the weight ρ to $\rho' (> \rho)$ by relocating one of the center locations, which then uncovers m' red points and one blue point (whose weight is, say, $-|\mathcal{R}|\delta + \epsilon$ for some $\epsilon > 0$). If we do so, then the updated weight would be $\rho' = \rho - m'\delta + |\mathcal{R}|\delta - \epsilon$. But, ρ' is no better than the earlier weight ρ since $m' \leq |\mathcal{R}|$, $\delta < 0$ and $(-m'\delta + |\mathcal{R}|\delta - \epsilon) < 0$. Further, the optimal solution with total weight ρ for the CSOFL (computed by using the algorithm of Theorem 4.11) is also optimal for this particular variant since ρ can not be improved by uncovering only red points. Hence, the above-proposed algorithm for the CSOFL problem will also correctly solve the ALLBLUE-MINRED problem. \square

Claim 4.16. The algorithm of Theorem 4.11 solves the MAXBLUE-NORED problem optimally.

Proof. Consider an instance of the CSOFL problem, in which every $p_i \in \mathcal{B}$ is associated with the weight $w_i = \delta$, and every $p_i \in \mathcal{R}$ is associated with the weight $w_i < -|\mathcal{B}|\delta$, where $\delta \in \mathbb{R}$ and $\delta > 0$.

Feasibility: Consider the following trivial feasible solution for the CSOFL problem. Let us place k facility center locations on ℓ so that they don't cover any blue points. Further, we reduce their radius so that no red points lie in the interior. Note that the total weight of the demand points covered by these facilities is zero. Hence, these k center locations form a feasible solution for the MAXBLUE-NORED problem since none of the red points are covered, and the total weight is zero.

Optimality: Suppose we have a feasible solution with a total weight ρ for the CSOFL problem. We will try to increase this weight by relocating one of the centers covering additional n' blue points and (at least) one red point. The update weight would be $\rho' = \rho + n'\delta - |\mathcal{B}|\delta$ which is smaller than ρ since $n' \leq |\mathcal{B}|$ and $\delta > 0$. Hence, we cannot improve the total weight by perturbing some centers to cover one more red point with the hope that it may allow us to cover some more (or even all) blue points. When we have an optimal solution with the total weight ρ for an instance of the CSOFL problem, the weight the covered blue points contribute can not be increased due to its optimality. On the other hand, this solution also can not cover any red point because we can get a better weight $\rho' = \rho - n'\delta + |\mathcal{B}|\delta$ (by reducing the radius to uncover these

red points. While doing so, we possibly uncover some blue points, say n' .) This would contradict that ρ is the optimum. Hence, the above-proposed algorithm (of Theorem 4.11) for the CSOFL problem will also correctly solve the MAXBLUE-NORED problem. \square

Corollary 4.17. *The ALLBLUE-MINRED and MAXBLUE-NORED problems can be solved in $O(n^3k \cdot \max(\log n, k))$ time.*

Proof. Since there are only two types of weights (namely, δ and $|\mathcal{B}|\delta$ or $-|\mathcal{R}|\delta$), instead of answering circular range reporting queries, we answer range counting queries [13], viz. blue count and red count for each of $O(nk)$ query circles of every candidate radius. Hence the total running time is $n^2 \cdot (O(nk^2) + O(nk \log n)) = O(n^3k \cdot \max(\log n, k))$. Hence, the theorem follows from Theorem 4.11 and Claims 4.15 and 4.16. \square

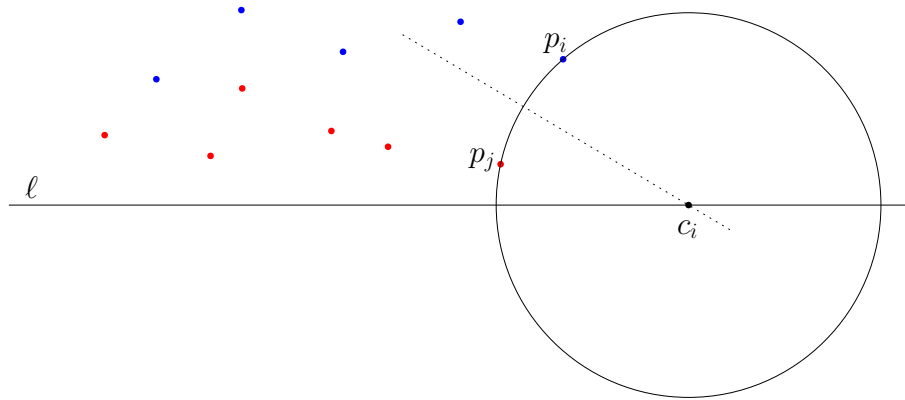
4.4.1 The MAXBLUE-NORED problem for $k = 1$

In this section, we address the problem of determining the minimum enclosing disk with center on ℓ , which encloses the maximum number of blue points without enclosing any red points. Recall that in the MAXBLUE-NORED problem, we are given two sets of points, blue points \mathcal{B} and red points \mathcal{R} , lying above a horizontal line ℓ , where $|\mathcal{B}| + |\mathcal{R}| = n$, the goal is to compute a minimum enclosing disk that maximizes the count of blue points (being enclosed in that disk) while ensuring that no red point is enclosed.

Observation 4.18. If the perpendicular bisector of any two points p_i and p_j intersects ℓ at c_i , then there exists a disk centered at c_i which has p_i and p_j on its boundary.

The method for solving the problem is as follows:

- For each pair of points in the set $\mathcal{B} \cup \mathcal{R}$, compute the perpendicular bisector of the line segment connecting them. Store the intersection points of these perpendicular bisectors with the line ℓ in a set I . Also, add to the set I all intersection points of ℓ with a vertical line through each blue point since only one blue point may also lie on the boundary of a disk in the optimal solution.
- For each $p_i \in I$, construct a disk centered at p_i that passes through the pair of points for which the perpendicular bisector was computed in the previous step.
- For each disk centered at $p_i \in I$, determine whether it contains any point from the set \mathcal{R} . If so, remove p_i from the set I . Otherwise, compute the number of blue points contained in the disk.
- If $|I| = 0$, then there exists no feasible solution. Otherwise, among the disks centered at points in I , select the one that contains the maximum number of blue points.

FIGURE 4.8: The optimal solution with only one blue point for $k = 1$.

Theorem 4.19. *The MAXBLUE-NORED problem for $k = 1$ can be solved in $O(n^3)$ time.*

Proof. In order to compute I , it requires $O(n^2)$ time. If all points in the set $\mathcal{B} \cup \mathcal{R}$ have distinct x -coordinates, then $|I| = O(n^2)$. Next, for each point $p_i \in I$, the time required to check the interiority of points is $O(n)$. Therefore, the total time complexity of the algorithm is $O(n^2) + O(n^3) = O(n^3)$. \square

Improved algorithm: Here, we improve the running time of the algorithm of Theorem 4.19 by almost a linear factor. Let us recall the notations, for a point p , we denote its coordinates by (x_p, y_p) , and for a pair of points p, q , we let $C_{p,q}$ denote the circle whose center lies on the line $y = 0$ and whose boundary passes through p, q , and we let $(x_{p,q}, 0)$ denote the center of $C_{p,q}$. Let C_p be a circle with its boundary passing through a point p , and its center lying on ℓ such that its radius equals y_p and its center at the coordinates $(x_p, 0)$.

Claim 4.20. Given three points p, q, r , the point r lies on or inside $C_{p,q}$ if and only if one of the following is true:

(i) $x_p < x_r$ and $x_{p,q} \geq x_{p,r}$;

(OR)

(ii) $x_p > x_r$ and $x_{p,q} \leq x_{p,r}$.

Proposition 4.21. *There is an algorithm that accepts two sets \mathcal{B}, \mathcal{R} of n ($|\mathcal{B}| + |\mathcal{R}|$) points on the plane and finds for every pair p, q of points in \mathcal{B} , the number of points of \mathcal{R} that lie on or inside the circle $C_{p,q}$. Further, this algorithm runs in time $O(n^2 \log n)$.*

Proof. We first describe the algorithm.

1. Sort the point sets $\mathcal{B} \cup \mathcal{R}$ based on their x -coordinates from left to right.

2. For each $p \in \mathcal{B}$, compute three lists: $C_{p,\mathcal{B}} = \{x_{p,q} | q \in \mathcal{B} \setminus \{p\}\}$, $C_{p,\mathcal{R},1} = \{x_{p,r} | r \in \mathcal{R} \text{ and } x_p < x_r\}$, $C_{p,\mathcal{R},2} = \{x_{p,r} | r \in \mathcal{R} \text{ and } x_p > x_r\}$.
3. For each p , sort the lists $L_{p,1} = C_{p,\mathcal{B}} \cup C_{p,\mathcal{R},1}$ and $L_{p,2} = C_{p,\mathcal{B}} \cup C_{p,\mathcal{R},2}$.
4. For each p , do the following: by making a single pass over $L_{p,1}$, compute for every $q \in \mathcal{B} \setminus \{p\}$, the value $N_{p,q,1}$, which is defined to be the number of elements of $C_{p,\mathcal{R},1}$ that appear before $x_{p,q}$ in $L_{p,1}$.
5. For each p , compute for every $q \in \mathcal{B} \setminus \{p\}$, the value $N_{p,q,2}$, which is defined to be the number of elements of $C_{p,\mathcal{R},2}$ that appear after $x_{p,q}$ in $L_{p,2}$.
6. For each p, q , the desired count (i.e., the number of red points covered by the disk $C_{p,q}$) is $N_{p,q,1} + N_{p,q,2}$.
7. To examine the scenario where only a single blue point resides on the circle, we construct a list in the following manner:
 - For each $p \in \mathcal{B}$, we select an arbitrary point p_{temp} that lies on the circle C_p .
 - Let $C_{\mathcal{B}} = \{(p, p_{temp}) | p \in \mathcal{B} \text{ and } p_{temp} \text{ is an arbitrary point lying on } C_p\}$ be a list.
 - Now, assign $C_{p,\mathcal{B}} = \{x_{p,q} | (p, q) \in C_{\mathcal{B}}\}$ in step 2 and compute the lists $C_{p,\mathcal{R},1}$ and $C_{p,\mathcal{R},2}$, then repeat the remaining steps till step 6.
8. Lastly, we determine the circle that encloses the maximum number of blue points and none of the red points by answering circular range counting queries for every circle $C_{p,q}$ which has the red count $N_{p,q,1} + N_{p,q,2} = 0$

Analysis: The correctness follows from Claim 4.20. The running time is dominated by steps 3, 4, 5, and 8. Step 3 takes time $O(n \log n)$ for a single point p and hence total time $O(n^2 \log n)$; steps 4 and 5 take time $O(n^2)$ each. The time complexity of Step 8 is $O(n^2 \log n)$ due to the repetition of the algorithm to determine the maximum number of blue points (i.e., the value $N_{p,q,1} + N_{p,q,2}$ is maximum for the blue points) enclosed by each circle $C_{p,q}$ satisfying $N_{p,q,1} + N_{p,q,2} = 0$ for the points in \mathcal{R} .

□

Theorem 4.22. *The MAXBLUE-NORED problem for $k = 1$ can be solved in $O(n^2 \log n)$ time.*

4.5 SOFL on t -lines

Let us consider a given set \mathcal{B} of blue points and a set \mathcal{R} of red points, which are positioned around t parallel lines denoted as $\ell_1, \ell_2, \dots, \ell_t$ in the plane. These lines may have arbitrary

vertical displacements. Each point $p_i \in \mathcal{B} \cup \mathcal{R}$ is assigned a weight denoted as w_i , where $w_i > 0$ if $p_i \in \mathcal{B}$ and $w_i < 0$ if $p_i \in \mathcal{R}$. The cardinality of the set $\mathcal{B} \cup \mathcal{R}$ is denoted as n , and the interior of any geometric object d is represented as d^0 (excluding its boundary ∂d).

The objective is to pack k non-overlapping congruent disks, denoted as d_1, d_2, \dots, d_k , with the smallest possible radius. These disks must be centered on the parallel lines closest to the points covered by each disk. The goal is to maximize the sum of the weights of the points covered by the interior of the disks. This sum is represented as $\sum_{j=1}^k \sum_{i: \exists p_i \in \mathcal{R}, p_i \in d_j^0} w_i + \sum_{j=1}^k \sum_{i: \exists p_i \in \mathcal{B}, p_i \in d_j} w_i$.

We may consider this as a generalization of the SOFL problem, where t horizontal lines are present, and facilities can be centered on any of these lines. Following a similar approach as in Section 4.2, we obtain all the candidate radii \mathcal{L}_{CAN} independently for each of the t lines and let us denote it as $\mathcal{L}_{\text{TCAN}}$. Note that the cardinality of $\mathcal{L}_{\text{TCAN}}$ is $O(tn^2)$. Hence, we have the following lemma.

Lemma 4.23. $|\mathcal{L}_{\text{TCAN}}| = O(tn^2)$

Next, we fix a radius $r_{\text{CAN}} \in \mathcal{L}_{\text{TCAN}}$. We can transform the problem into finding the minimum weight k -link path in a directed acyclic graph (DAG) $G(V', E')$, as discussed in Section 4.3. However, the cardinality of the set V' is $O(nkt^2)$, since each point $p_i \in \mathcal{B} \cup \mathcal{R}$ can create an influence interval on each of the t lines, resulting in $O(nt)$ endpoints of the influence intervals and adding $O(kt)$ additional points (see Figure 4.9). Figure 4.9 depicts the candidate locations on ℓ_{i+1} and ℓ_{i-1} , located at a distance of 2λ to the right of p_{ℓ_i} . Similarly, the mirror case can be considered for the point situated at a distance of 2λ to the left of p_{ℓ_i} on ℓ_{i+1} and ℓ_{i-1} .

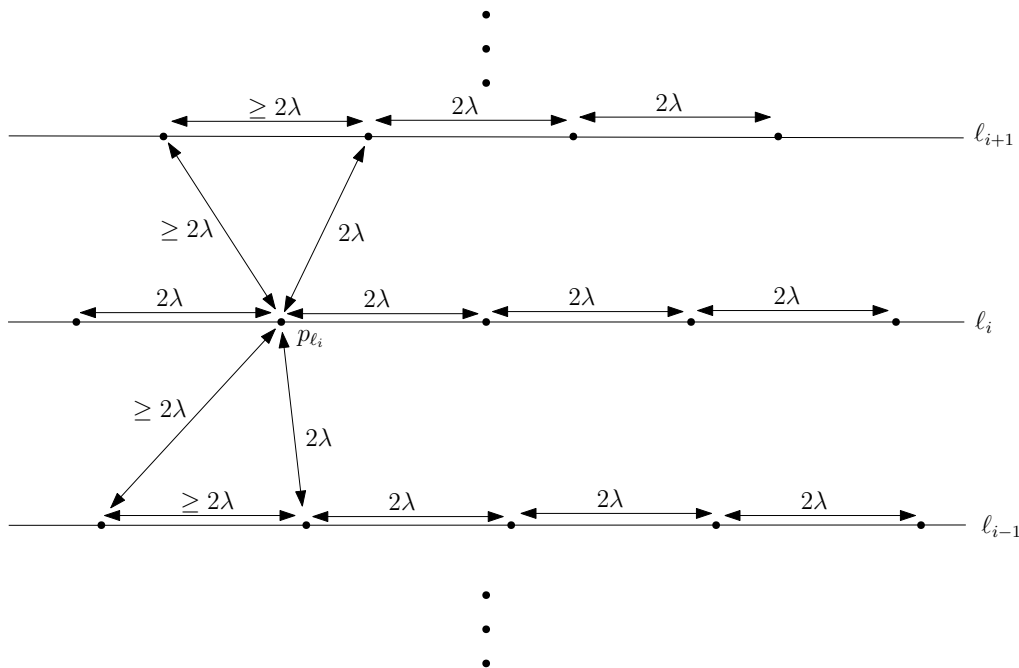


FIGURE 4.9: Candidate locations corresponding to the endpoint of the infeasible region p_{ℓ_i} and candidate radius λ .

Without loss of generality, we assume that all the points in V' have distinct x -coordinates. We can construct G' in $O(n^2k^2t^4)$ time by employing the sweeping technique, specifically sweeping from left to right. Therefore, the following lemma holds.

Lemma 4.24. *The DAG G' on t -lines can be constructed in $O(n^2k^2t^4)$ time.*

Proof. Follows from the Lemma 4.8 as the cardinalities $|V'| = O(nkt^2)$ and $|E'| = O(n^2k^2t^4)$. \square

Theorem 4.25. *The SOFL problem of t -lines can be solved exactly in $O(n^4k^2t^5)$ time.*

Proof. Follows from the Lemma 4.23 and Lemma 4.24 since there are $O(n^2t)$ candidate radii and the total time is $O(n^2t) \times O(n^2k^2t^4) = O(n^4k^2t^5)$. \square

4.6 Discrete SOFL with all facility sites in convex position

Suppose we are given a set \mathcal{B} of blue points, a set \mathcal{R} of red points, and a set \mathcal{F} of s candidate locations in convex position; all these three sets are in the plane. Let the weight of a given point $p_i \in \mathcal{B} \cup \mathcal{R}$ be $w_i > 0$ if $p_i \in \mathcal{B}$ and $w_i < 0$ if $p_i \in \mathcal{R}$, $|\mathcal{B} \cup \mathcal{R}| = n$, and $d^0 (= d \setminus \partial d)$ be the interior of any geometric object d . We wish to pack k non-overlapping congruent disks d_1, d_2, \dots, d_k of minimum radius, centered at points in \mathcal{F} such that $\sum_{j=1}^k \sum_{\{i: \exists p_i \in \mathcal{R}, p_i \in d_j^0\}} w_i + \sum_{j=1}^k \sum_{\{i: \exists p_i \in \mathcal{B}, p_i \in d_j\}} w_i$ is maximized, i.e., the sum of the weights of the points covered by $\bigcup_{j=1}^k d_j$ is maximized.

The above problem is a discrete variation of the SOFL problem (DSOFL) because a finite number of candidate facility sites (in convex position) are pre-given. Even though it is the discrete version of the SOFL problem, similar to the continuous line case, we know that there exists only a constant number of critical configuration types for the points in $\mathcal{R} \cup \mathcal{B}$ and candidate facilities in \mathcal{F} . From the latter, we also have a finite number of candidate radii here. Let $\mathcal{L}_{\text{DCAN}}$ denote the set of all candidate radii.

Lemma 4.26. $|\mathcal{L}_{\text{DCAN}}| = O(ns)$.

Proof. Since there will be only a constant number of critical configuration types concerning points $\mathcal{B} \cup \mathcal{R}$ and candidate facilities \mathcal{F} , we can consider the following situation where the candidate radius is determined based on a point in $\mathcal{B} \cup \mathcal{R}$ and a candidate facility location (on whose boundary that point lies) in \mathcal{F} . The cardinality of the set of radii from this situation is $O(ns)$ (see Figure 4.10).

The radius of the disks in the optimal packing cannot be determined solely by the distance between the candidate sites (see Fig 4.11). In Figure 4.11, we can observe that the closest pair

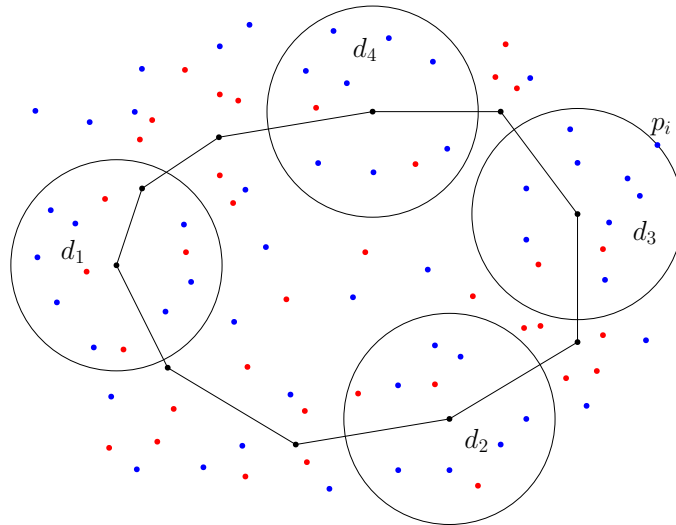


FIGURE 4.10: The blue point p_i lying on the boundary of d_3 will determine the radius.

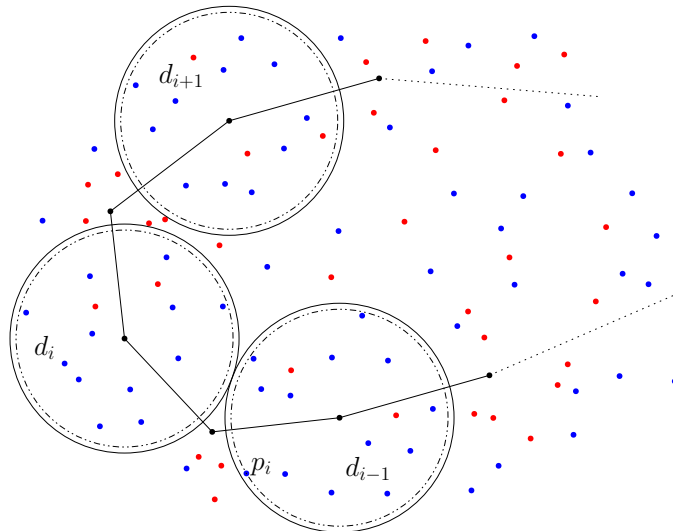


FIGURE 4.11: Illustration of the candidate facilities will not determine the radius of disks in the optimal packing.

of disks d_{i-1} and d_i will never touch in any optimal packing (i.e., the distance between them will not determine the radii of the disks in the optimal packing). Suppose they touch in any optimal packing, then we can reduce the radii of the disks until one of the blue points lies on the boundary of any of the disks (see Figure 4.11, p_i lying on the boundary of the disk d_{i-1}). \square

4.6.1 Dynamic programming algorithm

In this section, first, we show a relationship between the Voronoi diagram of points in an optimal solution and the cost of an optimal solution to the DSOFL problem. We then present a dynamic programming-based solution for the discrete SOFL problem utilizing this property of the Voronoi diagram (\mathcal{VD}) of the k sites in an optimal solution. This process is repeated for each $\lambda \in \mathcal{L}_{\text{DCAN}}$.

The Voronoi diagram of points in convex position forms a tree-like structure except for its infinite edges. If we overlay the diagram with a sufficiently big bounding rectangle, we have the following observation.

Observation 4.27. The Voronoi diagram of points in convex position is a tree.

Since the points in \mathcal{F} are in convex position, Observation 4.27 implies that the \mathcal{VD} of any subset of points in \mathcal{F} is also a tree. Hence, \mathcal{VD} of the optimal k facility sites is a tree. This \mathcal{VD} tree structure allows us to employ dynamic programming. To find this tree or a subtree of it from its rightmost node, we define a subproblem that explores all possible edges from the rightmost node and then further this exploration recursively. This leads to recursively constructing an optimal solution once we guess the rightmost node of the tree. Furthermore, we show no circular dependencies between subproblems.

Without loss of generality, let us consider that the points in $\mathcal{F} = \{p_1, p_2, \dots, p_s\}$ are ordered clockwise. It is known that the Delaunay triangulation is the dual of the Voronoi diagram. Denote \mathcal{DT} as the Delaunay triangulation formed by the points corresponding to the Voronoi centers in the Voronoi diagram, \mathcal{VD} . Observe that the smallest edge length of \mathcal{DT} of points in an optimal solution to DSOFL is at least twice the radius of disks in the optimal solution.

For a given $\lambda \in \mathcal{L}_{\text{DCAN}}$, we precalculate the weight of points covered by a disk with radius λ centered at a facility site $f_i \in \mathcal{F}$ and denote this weight as $w(f_i)$. Then, our dynamic program-based algorithm is as follows. First, we guess the Delaunay triangle corresponding to the rightmost Voronoi node, the three rightmost facility centers, say, p_i, p_ℓ, p_j , (where p_ℓ is the rightmost and p_i is below p_j) in the optimal solution. We make all possible $\binom{s}{3}$ guesses to find these three optimal centers. Then, define a subproblem $\Gamma(p_i, p_\ell, p_j, \mathcal{F}'; \mathcal{K})$, which corresponds to the maximum (optimal) weight of the points covered by \mathcal{K} facilities located at some points in \mathcal{F} with a radius of λ , and the points p_i, p_ℓ and p_j are the rightmost ordered points in the optimal solution. Initially, we set $\mathcal{K} = k - 3$ and $\mathcal{F}' = \mathcal{F} \setminus \{p_i, \dots, p_\ell, \dots, p_j\}$, where the indices i, j, ℓ, ℓ' are to be read modulo s . Now, consider reconstructing \mathcal{VD} with three p_i, p_j, p_ℓ fixed on the right. We do this by determining the corresponding \mathcal{DT} triangle with its corner points p_i, p_j and $p_{\ell'}$. We extend the \mathcal{VD} by choosing the next point $p_{\ell'}$ that lies left of $\overrightarrow{p_j p_i}$ such that it is at least 2λ from p_i, p_j and p_ℓ . Observe that $p_{\ell'}$ is outside the circumcircle of p_i, p_j and p_ℓ . Then we have the following recurrence,

$$\Gamma(p_i, p_\ell, p_j, \mathcal{F}'; \mathcal{K}) = \begin{cases} \max_{\substack{\mathcal{K}' \leq \mathcal{K} - 1, \\ p_{\ell'} \in \mathcal{F}' \text{ and} \\ p_{\ell'} : \zeta(p_{\ell'}, \{p_i, p_j, p_\ell\}) \geq 2\lambda}} \left\{ \begin{array}{l} w(p_{\ell'}) + \Gamma(p_{\ell'}, p_i, p_j, \mathcal{F}''; \mathcal{K}') + \Gamma(p_{\ell'}, p_j, p_i, \mathcal{F}'''; \mathcal{K} - 1 - \mathcal{K}') \\ 0 \end{array} \right. & \text{if } |\mathcal{F}'| \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

where $\zeta(p_{\ell}, \{p_i, p_j, p_{\ell}\}) = \min\{\text{dist}(p_{\ell}, p_i), \text{dist}(p_{\ell}, p_j), \text{dist}(p_{\ell}, p_{\ell})\}$, $w(p_{\ell})$ denotes the total weight of the points that are covered by a disk of radius λ centered at p_{ℓ} , $\mathcal{F}'' = \mathcal{F}' \setminus \{p_j, \dots, p_{\ell}\}$ and $\mathcal{F}''' = \mathcal{F}' \setminus \{p_{\ell}, \dots, p_i\}$. Base cases are $\Gamma(p_i, p_{\ell}, p_j, \mathcal{F}'; 0) = w(p_i) + w(p_{\ell}) + w(p_j)$, $\Gamma(p_i, p_{\ell}, p_j, \emptyset; \mathcal{K}) = 0$.

Proof of Correctness:

The correctness of the dynamic programming algorithm can be established based on the following observations:

- The minimum edge length of \mathcal{DT} formed by the k sites in the optimal solution is at least 2λ . This ensures that the disks in the optimal solution do not overlap, as the distance between any two points in the solution is greater than or equal to 2λ .
- There always exists a solution for a given set of points $\mathcal{B} \cup \mathcal{R}$ and \mathcal{F} . If it is impossible to place k disks with a radius of λ , the algorithm returns a zero weight, indicating that a solution does not exist for given λ .
- By assuming that p_i , p_{ℓ} , and p_j are the rightmost points in the optimal solution, we have $O(s^3)$ choices for these points. This assumption ensures that a \mathcal{DT} with k vertices corresponding to the optimal solution always exists if a solution exists for a given λ .

Based on these observations, we can conclude that the dynamic programming algorithm is correct in determining the optimal solution for the given set of points $\mathcal{B} \cup \mathcal{R}$ and \mathcal{F} , considering the assumptions made and the properties of the \mathcal{DT} formed by the candidate sites.

Theorem 4.28. *Discrete SOFL with candidate facility sites in convex position can be solved in polynomial time.*

Proof. The running time of the algorithm is calculated as follows:

- From Lemma 4.26 we have $|\mathcal{L}_{\text{DCAN}}| = O(ns)$.
- For each $\lambda \in \mathcal{L}_{\text{TCAN}}$ we call the dynamic programming algorithm.
- Dynamic programming algorithm for a given λ :
 - For each $f_i \in \mathcal{F}$, calculating weight of points covered by a disk of radius λ centered at f_i will take (ns) time.
 - There are $O(s^3k)$ subproblems and each subproblem will take $O(sk)$ time.

- The total time complexity of the algorithm is $O(n^2s^2 + ns^5k^2)$. Additionally, we designate the vertices of \mathcal{DT} as the optimal solution that yields the maximum weight out of all the invocations of the dynamic programming algorithm with three rightmost points p_i, p_j, p_ℓ .

□

4.7 Conclusion

In this chapter, we have studied the SOFL problem on a line and provided an exact algorithm that runs in $O(n^4k^2)$ time. Furthermore, we have presented an improved dynamic programming-based solution with a time complexity of $O(n^3k \cdot \max(n, k))$. Moreover, we have investigated two specific cases of the CSOFL problem, which involve only two sets of weighted points. We showed that these two problems can be solved in $O(n^3k \cdot \max(\log n, k))$ time. Additionally, we have devised faster algorithms for the MAXBLUE-NORED problem when $k = 1$. Finally, we extend the result of SOFL to t -lines and for the points in convex position and showed that these can also be solved in polynomial time.

The results in this chapter are submitted for publication [73].

Chapter 5

Max-Min k -Dispersion for Points in Convex Position in the Plane

In this chapter, we consider the max-min k -dispersion problem for points in convex position as follows:

Discrete k -dispersion on a Convex Polygon (DKCONP): Given a set S of n points in convex position, assume that the points in S are ordered in a clockwise order around the centroid of S , forming a convex polygon \mathcal{P} . Then, observe that the k -dispersion problem on the set S can be equally stated as packing k congruent disks of maximum radius, with their centers lying at the vertices of the convex polygon \mathcal{P} .

For this problem, we propose two exact algorithms for the DKCONP problem here. First, we propose an FPT algorithm. The running time of this algorithm is $O(2^k n^2 \log^2 n)$, which is an improvement over the previous best exact algorithm (running in $n^{O(\sqrt{k})}$) [5]. Here, the constant hidden in $O(\sqrt{k})$, the exponent of n in the running time, is larger than 5.44 [51]. Secondly, we give an exact polynomial time algorithm based on dynamic programming in $O(n^4 k^2)$ time for any $k > 0$. For $k = 3$, the existing (exact) algorithm runs in $O(n^2)$ time [44]. For small values of k the FPT algorithm is faster than the proposed polynomial time algorithm. Finally, we give a linear time $\frac{1}{2\sqrt{2}}$ -approximation algorithm for $k = 3$.

5.1 Preliminaries

This section introduces some terminologies and observations useful in discussing our solution for the DKCONP problem.

Let us use $\overline{v_i v_j}$ to denote the line segment connecting the points v_i and v_j . We use $|\cdot|$ (i) to denote the length $|v_i v_j|$ of the line segment $\overline{v_i v_j}$, (ii) to denote the absolute value $|x|$ of a real number $x \in \mathbb{R}$, and also, (iii) to denote the cardinality $|S|$ of any set S . The center of any disk d is denoted by $\mathcal{C}(d)$, and the diameter of any convex polygon \mathcal{P} is denoted by $\mathcal{D}(\mathcal{P})$. Let r_{max} be the (maximum) radius of the disks in an optimal solution to the DKCONP problem. Let $\overline{v_i v_j}$ be a chord of \mathcal{P} corresponding to the pair (v_i, v_j) of vertices of \mathcal{P} , where $1 < |i - j| < n - 1$ and $i, j \in \{1, 2, \dots, n\}$. Let $C = \{\overline{v_i v_j} : 1 < |i - j| < n - 1, i, j \in \{1, 2, \dots, n\}\} \cup \{\overline{v_1 v_2}, \overline{v_2 v_3}, \dots, \overline{v_n v_1}\}$ be the set of chords and edges of \mathcal{P} , where $\overline{v_i v_{i+1}}$, for $i = 1, 2, \dots, n - 1$, are the edges of \mathcal{P} and $\overline{v_n v_{n+1}} = \overline{v_n v_1}$. Clearly, $|C| = \frac{n(n-1)}{2}$. Now, let $C' = \{|v_i v_j| \mid i, j = 1, 2, \dots, n \text{ and } \overline{v_i v_j} \in C\}$ be the set of all distinct distances between pairs of vertices of \mathcal{P} .

Observation 5.1. $2r_{max} \in C'$ and $|C'| = O(n^2)$.

Due to Observation 5.1, we can find r_{max} in at most $\lceil 2 \log n \rceil$ stages of the binary search, provided that for any given r we can decide whether $r > r_{max}$ or $r \leq r_{max}$. Based on a bounded search tree technique, we propose a fixed-parameter algorithm to answer this decision question, where k is the parameter.

5.2 An exact fixed-parameter algorithm

For the DKCONP problem, here we aim to develop a fixed-parameter algorithm using the bounded search tree method. First, we consider the decision version of the DKCONP problem for a given candidate radius r and then provide an algorithm to solve it. This algorithm is based on a 2-way search tree, the depth of which is k . Therefore, the tree contains at most $O(2^k)$ nodes in total, and at each of these nodes, we attempt to place a disk centered at some given point. Next, we discuss the optimization scheme, where we need to find the maximum radius r_{max} for which this decision algorithm returns YES. To this end, we first pre-compute all the candidate radii, as they are finite in number and are half of all pairwise distances between given points. The optimal value r_{max} will be one of these candidate radii. Using a linear time selection algorithm, we then find the median of the set of candidate radii. If the decision algorithm returns YES for the median radius, we discard all radii less than the median. If the decision algorithm returns NO, we discard all radii greater than the median. This process continues until only one element remains in the set of candidate radii, for which the decision algorithm returns YES and it is the optimal radius.

The decision algorithm is defined as follows:

DECISION(\mathcal{P}, k, r): Given a convex polygon \mathcal{P} with n vertices and a positive integer $k < n$ and a radius r , is it possible to pack k (non-overlapping) congruent disks of radius r , with centers lying at the vertices of \mathcal{P} ?

Observe that the answer to $\text{DECISION}(\mathcal{P}, k, r)$ is YES if the radius r is less than or equal to the radius r_{max} of the disks in an optimal solution of the DKCONP problem. Now, we shall design an algorithm that solves $\text{DECISION}(\mathcal{P}, k, r)$ in $O(f(k) \cdot n^{O(1)})$ time and returns a set of k disks of radius r packed on the boundary of \mathcal{P} if the answer is YES, and returns NO otherwise, where $f(k)$ is an arbitrary exponential function in k .

5.2.1 Decision algorithm

The outline of the algorithm is as follows. First, we align the polygon \mathcal{P} such that its leftmost vertex v_1 is placed at the origin. Then, we place the disk d_1 of radius r centered at v_1 . From the vertex v_1 in a clockwise (CW) direction along the boundary of \mathcal{P} , we find the first vertex u at which we can center a r -radius disk d_2 that does not overlap with any previously placed disks. Now, we again have two ways to place the next disk d_3 , namely, moving in a clockwise direction from the center of d_2 or moving in a counter-clockwise (CCW) direction from the center of d_1 . Similarly, from the vertex v_1 in a counter-clockwise direction along the boundary of \mathcal{P} , we find the first vertex u' at which we could center the disk d_2 . In this way, our search for finding all $k - 1$ vertices of \mathcal{P} (as the centers) to pack the disks proceeds like a 2-way search tree (see Figure 5.1 for $k = 4$). The depth of the search tree is k because we stop after placing k disks and return the disks. At any point along a path of the 2-way search tree, if we can not place a disk, we backtrack to placing a disk in the other direction. Thus, the branching factor of every node is at most 2, resulting in $O(2^k)$ nodes in total. We repeat the above procedure by placing the disk d_1 at each of the n vertices of \mathcal{P} . Note that the disks corresponding to the vertices of any path of length $\geq k$ in the 2-way search tree together form a feasible solution for the DKCONP problem.

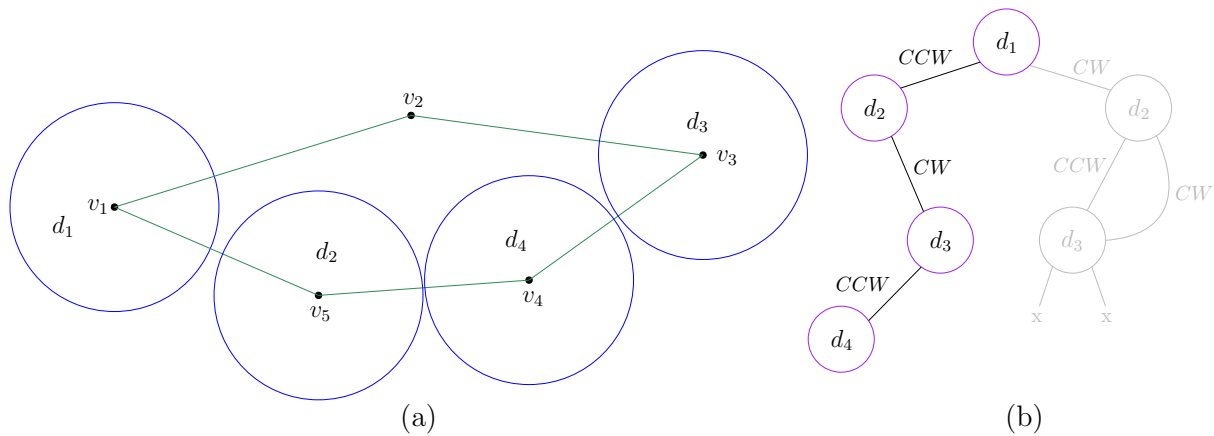


FIGURE 5.1: (a) Disks placed by the decision algorithm for $k = 4$ (b) Corresponding 2-way search tree.

Now, we shall describe how to pack the next disk d_{j+1} after having packed the disks d_1, d_2, \dots, d_j . In the above 2-way search tree procedure, for the value of $j = 1, 2, \dots, k - 1$, after packing the disk d_j centered at some vertex of \mathcal{P} , the candidate vertices u and u' for the center of the

disk d_{j+1} can be computed as follows: let u be the first vertex at a distance of at least $2r$ from the center of the most recently packed disk (d_j or d_{j-1}) clockwise from the center of d_1 . Similarly, let u' be the first vertex at a distance at least $2r$ from the center of the most recently packed disk (d_j or d_{j-1}) counter-clockwise from the center of d_1 . Note that u and u' are the two candidate vertices for packing the next disk d_{j+1} , which will be centered at one of them. However, it is required to ensure that the distance between the candidate center vertex u or u' and each of the vertices at which the already-packed disks d_1, d_2, \dots, d_j are centered is at least $2r$. Observe that for a convex polygon \mathcal{P} , the distances between a fixed vertex and the remaining vertices of \mathcal{P} form a multi-modal function. Therefore, we can not directly employ binary search to find the center vertices u and u' for packing the next disk d_{j+1} , $j = 1, 2, \dots, k-1$. For a vertex v_i of \mathcal{P} , there are γ vertices that are the modes or local maxima [7], where $\gamma \leq n/2$. We will exploit this property to identify all the candidate center vertices and to quickly locate the one among them for centering the disk d_{j+1} . Hence, given a candidate radius r , we do some preprocessing before we shall call the decision algorithm.

5.2.2 The optimization scheme

To solve the optimization problem, i.e., to find the maximum value r_{max} of r , we solve $\text{DECISION}(\mathcal{P}, k, r)$ repeatedly while performing a binary search on C' . In each stage of the binary search, the radius r will be the median element of C' divided by 2. The median will be found using a linear-time median finding algorithm [18]. We then perform the above 2-way search tree-based procedure to find an answer to $\text{DECISION}(\mathcal{P}, k, r)$. If the answer is YES, then we update C' by removing all the elements of it that are smaller than $2r$. Otherwise, we update by removing all the elements that are at least $2r$. In either case, the size of the updated C' will be half of the previous C' . The main routine of the algorithm is outlined in Algorithm 4.

Preprocessing:

Here we describe how to precompute all the candidate center vertices for the next disk d_{j+1} ($j = 1, 2, \dots, k-1$) once an element $2r \in C'$ is fixed, where a candidate center vertex is a vertex of \mathcal{P} at which a disk is likely to be centered in the packing computed by Algorithm 4. We also see how to use this precomputed information in every $(j+1)$ th step of the decision algorithm given that the disks $\{d_1, d_2, \dots, d_j\}$ are packed on $\partial\mathcal{P}$ with centers $\mathcal{C}(d_1) = v_{\alpha_1}, \mathcal{C}(d_2) = v_{\alpha_2}, \dots, \mathcal{C}(d_j) = v_{\alpha_j}$, where $j = 1, 2, \dots, k-1$, and $\partial\mathcal{P}$ is the boundary of \mathcal{P} .

In Algorithm 4, we initially set $\mathcal{X}_1 = C'$, the set of all distances between the points in S . In the i th stage of the binary search (while loop in Algorithm 4), we have that $|\mathcal{X}_i| \leq \frac{|\mathcal{X}_{i-1}|}{2}$, where the set \mathcal{X}_i always contains the element $2r_{max}$ along with possibly some other candidate radii for $i = 2, 3, \dots, \lceil 2 \log n \rceil$. Now, consider a straight line ℓ_s (with any orientation) through any vertex

v_s of \mathcal{P} , that splits \mathcal{P} into two parts, each with at least one vertex other than v_s . Given a median $2r \in \mathcal{X}_i$, for each vertex v_s of \mathcal{P} the candidate center vertices $u_{s_1}, u_{s_2}, \dots, u_{s_\gamma}$ lying above any straight line ℓ_s through v_s (and $u_{s'_1}, u_{s'_2}, \dots, u_{s'_{\gamma'}}$ lying below ℓ_s) are such that for $1 \leq \beta \leq \gamma$ we have that $|v_s u_{s_{\beta-1}}| < 2r$, $|v_s u_{s_{\beta+1}}| > 2r$ and $|v_s u_{s_\beta}| \geq 2r$ or $|v_s u_{s_{\beta-1}}| > 2r$, $|v_s u_{s_{\beta+1}}| < 2r$ and $|v_s u_{s_\beta}| \geq 2r$, where $\max(\gamma, \gamma') \leq \frac{n}{2}$. These candidate vertices can be pre-computed by doing the distance checks while linearly scanning through $\partial\mathcal{P}$ both in clockwise and counter-clockwise from every vertex v_i . Hence, this pre-computation at the beginning of each stage (line 4) will take $O(n^2)$ time. The overall time across all stages of the binary search for these pre-computations will be $O(n^2 \log n)$. These candidate center vertices are stored in the array A_i for each vertex v_i (see line 4 of Algorithm 4). Let $v_{\alpha_{up}}$ be the vertex in clockwise order from v_{α_1} ($= \mathcal{C}(d_1)$), at which the recently packed disk is centered (see Figure 5.2). Let $v_{\alpha_{low}}$ be the vertex in counter-clockwise order from v_{α_1} , at which the recently packed disk is centered. Let $u_{i_1}, u_{i_2}, \dots, u_{i_\gamma}$ be the candidate center vertices in clockwise order from $v_{\alpha_{up}}$ for packing the next disk d_{j+1} . Similarly, the vertices $u_{i'_1}, u_{i'_2}, \dots, u_{i'_{\gamma'}}$ are the candidate center vertices in counter-clockwise order from $v_{\alpha_{low}}$.

Algorithm 4: Exact-fixed-parameter.

Input: A convex polygon \mathcal{P} with V vertices and an integer k

Output: Radius r_{max} of k disks packed

$\mathcal{X}_1 \leftarrow C'$, $i \leftarrow 1$

while $|\mathcal{X}_i| \geq 2$ **do**

$r \leftarrow \text{median}(\mathcal{X}_i)/2$ // invoke the linear-time median finding algorithm [18]
 Based on the value of $2r$, precompute the candidate center vertices for each $v_s \in V$ and store in a global array A_s , $s = 1, 2, \dots, n$.
if $DECISION(\mathcal{P}, k, r)$ **then**
 $\mathcal{X}_{i+1} \leftarrow \mathcal{X}_i \setminus \{e \in \mathcal{X}_i | e \leq 2r\}$
else
 $\mathcal{X}_{i+1} \leftarrow \mathcal{X}_i \setminus \{e \in \mathcal{X}_i | e \geq 2r\}$
 $i \leftarrow i + 1$

$r = \min(\mathcal{X}_i)/2$

return r

Computation of a center vertex for d_{j+1} by the decision algorithm:

Now consider the $(j + 1)$ th iteration in the i th stage of the binary search. Let us denote the right most disks in the packing $\{d_1, d_2, \dots, d_j\}$ on both upper and lower boundaries of $\partial\mathcal{P}$ by $d_{\alpha_{up}}$ and $d_{\alpha_{low}}$ centered respectively at $v_{\alpha_{up}}$ and $v_{\alpha_{low}}$ (see Figure 5.2). The candidate center vertices from the center of $d_{\alpha_{up}}$ in clockwise order are $u_{i_1}, u_{i_2}, \dots, u_{i_\gamma}$ and from the center of $d_{\alpha_{low}}$ in counter-clockwise order are $u_{i'_1}, u_{i'_2}, \dots, u_{i'_{\gamma'}}$. Merge these two lists into one single list in convex position order (i.e., respecting the initial given order in S) by discarding the candidate centers lying to the left of the line $\ell_{low,up}$ through $v_{\alpha_{low}}$ and $v_{\alpha_{up}}$ (see Figure 5.2). This merging will take $\gamma + \gamma' - 1 = O(n)$ time (as we need to check at most n vertices in the order of S). Observe that due to the convexity of \mathcal{P} each of the vertices $\mathcal{C}(d_1), \mathcal{C}(d_2), \dots, \mathcal{C}(d_j)$ lie either on

$\ell_{low,up}$ or to the left of $\ell_{low,up}$. Assume that the vertices (in clockwise order) between u_{i_1} and $u_{i'_{\gamma'-1}}$ all have distances at least $2r$ from both $v_{\alpha_{low}}$ and $v_{\alpha_{up}}$, and that $u_{i'_{\gamma'}}$ appears before u_{i_1} in clockwise order from $v_{\alpha_{up}}$. For each $p = i_1, i_1 + 1, \dots, i'_{\gamma'-1}$ consider the line $\ell_{p,up}$ through $v_{\alpha_{up}}$ and v_p , and the line $\ell_{p,low}$ through v_p and $v_{\alpha_{low}}$, respectively. Note that the distances from the line $\ell_{p,up}$ to the vertices $\mathcal{C}(d_1), \mathcal{C}(d_2), \dots, \mathcal{C}(d_j)$ satisfy unimodality because \mathcal{P} is in convex position. Similarly the distances from $\ell_{p,low}$ to $\mathcal{C}(d_1), \mathcal{C}(d_2), \dots, \mathcal{C}(d_j)$ satisfy unimodality property. Hence, we can use binary search to discard a vertex v_p if it is of distance strictly less than $2r$ from one of $\mathcal{C}(d_1), \mathcal{C}(d_2), \dots, \mathcal{C}(d_j)$, as follows: Let $\mathcal{C}(d_1), \mathcal{C}(d_2), \dots, \mathcal{C}(d_j)$ be in convex position order (in clockwise along $\partial\mathcal{P}$), find contiguous subsequences (if exists) of these points that have distance strictly less than $2r$ from the lines $\ell_{p,up}$ and $\ell_{p,low}$, by doing a binary search over the latter ordered list. Then check if there is a point in any of these sequences that is of distance strictly less than $2r$ from v_p , by using binary search again. Discard v_p if so. Also we linearly search the contiguous subsequence $u_{i_1}, u_{i_1+1}, \dots, u_{i'_{\gamma'-1}}$ to find the first vertex v_p clockwise from $v_{\alpha_{up}}$ such that $\mathcal{C}(d_{j+1}) = v_p$. This process takes $O(n + n \log j)$ amortized time, where $O(n)$ is for merging and $O(n \log j)$ for finding the center for the disk d_{j+1} by doing binary search on $\mathcal{C}(d_1), \mathcal{C}(d_2), \dots, \mathcal{C}(d_j)$. Similarly, we spend the same time if we are packing d_{j+1} in the counter-clockwise direction from $v_{\alpha_{low}}$. Then we have the following claim.

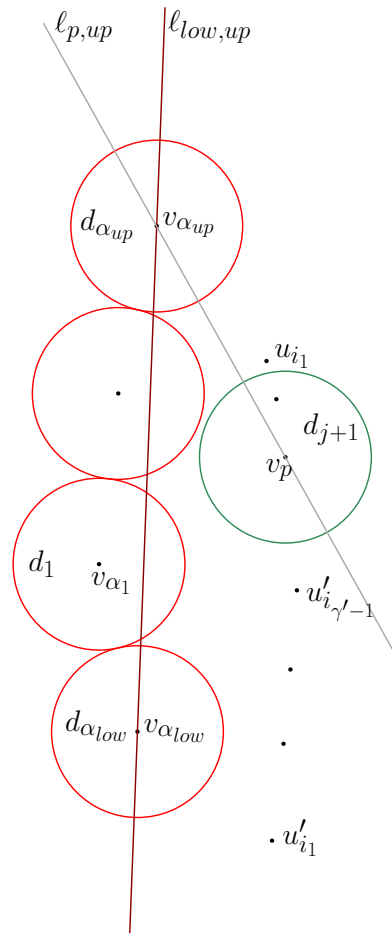


FIGURE 5.2: Preprocessing and computation of a center vertex for d_{j+1} .

Claim 5.2. If $\text{DECISION}(\mathcal{P}, k, r) = \text{YES}$ then there exists a vertex v_{α_1} of \mathcal{P} with $v_{\alpha_1} = \mathcal{C}(d_1)$ that results in the following: there is a root-leaf path of a 2-way search tree with the root corresponding to v_{α_1} such that at every node along the path we are able to pack the next disk d_{j+1} centered at one of the candidate vertices.

Proof. Suppose the disk d_{j+1} in the optimal packing is not centered at one of the candidate vertices in the $(j + 1)$ th iteration. Also, consider that previous j disks in the optimal packing are centered at the candidate vertices (let us call them optimal centers). Then it must be the case that the optimal center vertex at which d_{j+1} is centered (in the optimal packing) appears strictly between the first candidate vertices v_p and $v_{p'}$ (with distance at least $2r$ from each of $\mathcal{C}(d_1), \mathcal{C}(d_2), \dots, \mathcal{C}(d_j)$) respectively from $v_{\alpha_{up}}$ and $v_{\alpha_{low}}$. Otherwise, it should be one of the candidate vertices as the first vertices v_p and $v_{p'}$ are at the distance $2r$ from the centers of previously packed disks in clockwise and counter-clockwise directions, respectively. Now, we can perturb $\mathcal{C}(d_{j+1})$ to center d_{j+1} at the nearest candidate center without violating the packing property, and it also creates more space on $\partial\mathcal{P}$ (on the other side) for packing the following disks. Therefore, the claim follows by induction on j due to the above discussion (See Figure 5.2). □

Lemma 5.3. *We can answer the decision question $\text{DECISION}(\mathcal{P}, k, r)$ in $O(2^k n^2 \log n)$ time.*

Proof. The correctness of our decision algorithm follows due to the following facts:

1. Fix some vertex v of \mathcal{P} and a center of the disk d_1 at v . In the search space corresponding to the vertex v , along any (root-leaf) path (of the search tree) after the disk d_j is centered, by the claim above if $r \leq r_{max}$ there is always a candidate center vertex u in at least one direction along the boundary of \mathcal{P} in order to pack the next disk d_{j+1} , for $j = 2, 3, \dots, k - 1$. We argued that the amortized time for finding this candidate vertex is $O(n + n \log j)$ (by accessing the array A_s (for a vertex s) computed in step 4 of Algorithm 4). Hence, the branching factor of every node of the search space is at most 2. Therefore, after the disk d_1 is centered at some vertex v of \mathcal{P} , the resulting search space for the remaining $k - 1$ disks is a 2-way search tree, and its depth is $O(k)$.
2. Consider an element $2r \in C'$ such that $r \geq r_{max}$. Now, for this radius r let k' be the maximum number of disks that can be packed in the optimal packing OPT and $k \geq k'$. We can determine a vertex that is the center for the disk d_1^{OPT} in OPT , in $O(n)$ time by exploring the search space rooted corresponding to each vertex of the polygon \mathcal{P} . Then, by walking along $\partial\mathcal{P}$ from the point $\mathcal{C}(d_1^{\text{OPT}})$, we can charge each disk $d_{j'}^{\text{OPT}}$ with at least one disk d_j centered by Algorithm 4 if $(d_j \cap d_{j'}^{\text{OPT}}) \neq \emptyset$ for $j' = 1, 2, \dots, k'$. Therefore, $\mathcal{C}(d_1^{\text{OPT}}) = \mathcal{C}(d_1)$, i.e., d_1^{OPT} gets charged with d_1 itself. Suppose there is some optimal

disk $d_{j'}^{\text{OPT}}$ that does not get charged with any disk d_j centered by Algorithm 4. Then this will contradict the termination of Algorithm 4. This implies that $k = k'$.

3. In the 2-way search tree, there are at most 2^j nodes at level j . Since we invest $O(n + n \log j)$ time at every node of the level j , the total time will be $\sum_{j=1}^{k-1} (2^j (n + n \log j)) = O(2^k (n + n \log k))$.

If the radius $r \leq r_{\max}$, then we can answer $\text{DECISION}(\mathcal{P}, k, r)$ correctly in time $n \cdot (2^k (n + n \log k)) = O(2^k (n^2 + n^2 \log k)) = O(2^k n^2 \log n)$ since we exhaustively search all n 2-way search trees and $k \leq n$. \square

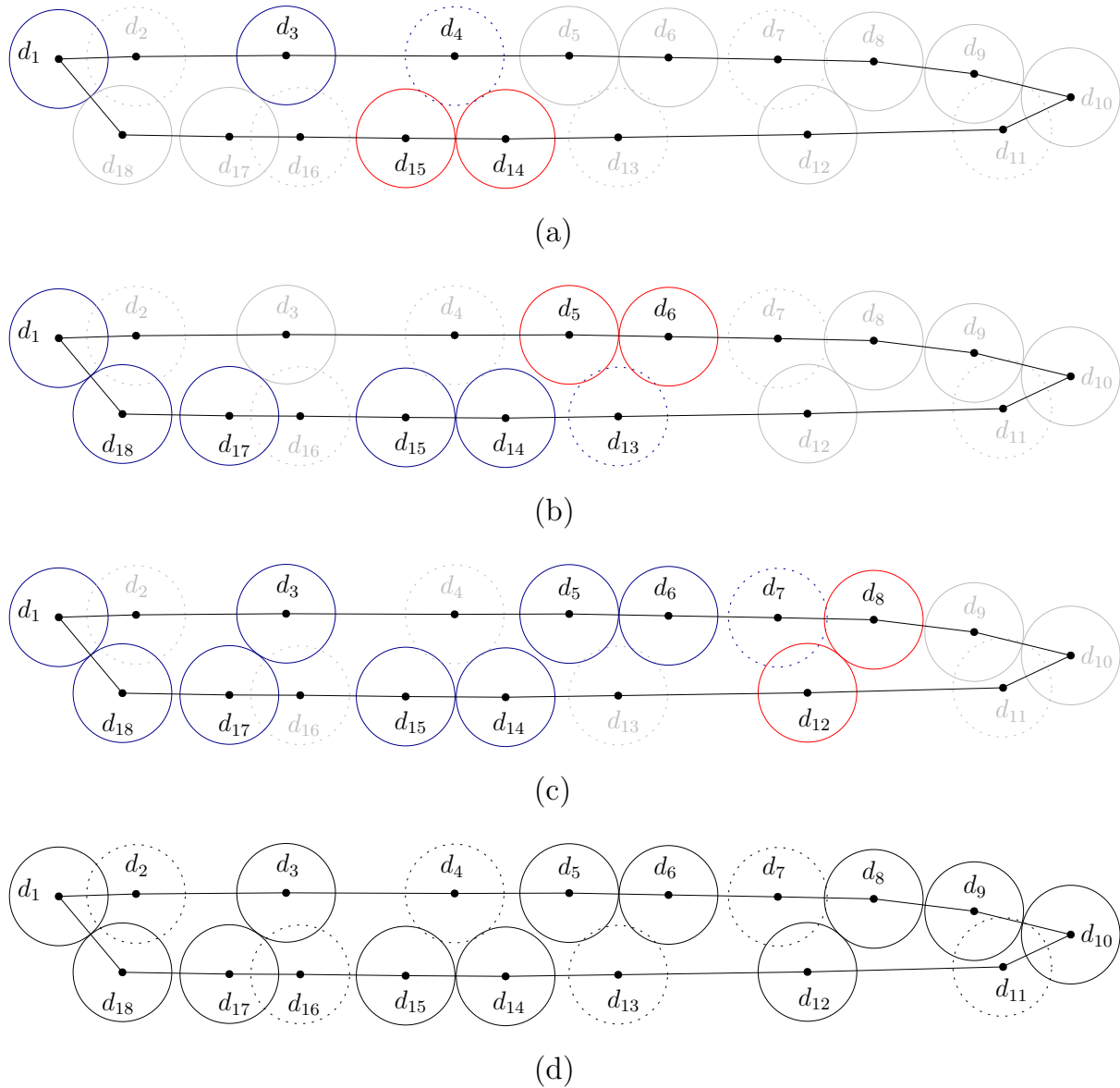
The other greedy strategies one can think of may not answer $\text{DECISION}(\mathcal{P}, k, r)$ correctly, as evident in the following discussion. This justifies that all possible (root-leaf) paths of the 2-way search tree must be explored to answer the decision question correctly.

For instance, let's take an example of a convex polygon with 18 vertices and $k = 12$, shown in Figure 5.3. In Figure 5.3, disk d_1 is the first disk placed at the leftmost point of the convex polygon. If we use the 2-way search tree in the $\text{DECISION}(\mathcal{P}, k, r)$, it returns the correct answer, and the disks in this packing for the path in the 2-way search tree are starting from d_1 , d_{18} in CCW , d_3 in CW , d_{17} in CCW , d_{15} in CCW , d_5 in CW , d_{14} in CCW , d_6 in CW , d_{12} in CCW , d_8 in CW , d_{10} in CCW , and d_9 in CW (see Figure 5.3 (d)).

Now, let's examine some alternative greedy strategies instead of the 2-way search tree and see how they fail for the above instance:

- **Greedy 1:** Place the next disk greedily by always choosing the first feasible point in the clockwise direction (see Figure 5.3 (a)). When the disk d_4 is placed with this strategy, we cannot place disks d_{14} and d_{15} , which are part of the optimal packing for $k = 12$.
- **Greedy 2:** Place the next disk greedily by always choosing the first feasible point in the counterclockwise direction (see Figure 5.3 (b)). Here, when the disk d_{13} is placed with this strategy, we cannot place disks d_5 and d_6 , which are part of the optimal packing for $k = 12$.
- **Greedy 3:** Place the next disk greedily by always choosing the first feasible point regardless of the clockwise or counterclockwise direction (see Figure 5.3 (c)). When the disk d_7 is placed with this strategy, we cannot place disks d_8 and d_{12} , which are part of the optimal packing for $k = 12$. Therefore, it's necessary to search every branch of the 2-way search tree to place the disks using the decision algorithm (see Figure 5.3 (d)).

Theorem 5.4. *We have an exact fixed-parameter algorithm for the $DKCONP$ problem in $O(2^k n^2 \log^2 n)$ time.*

FIGURE 5.3: (a) Greedy 1 (b) Greedy 2 (c) Greedy 3 (d) 2-way search tree for $k = 12$.

Proof. Follows from Lemma 5.3 and by doing a binary search on the set C' , since the total size of the search tree is bounded by a function of the parameter k alone, and every step takes polynomial time, and there are at most $\lceil 2 \log n \rceil$ calls to $\text{DECISION}(\mathcal{P}, k, r)$. \square

5.3 An exact polynomial time algorithm

In this section, we discuss an exact polynomial-time algorithm based on dynamic programming. This dynamic programming algorithm exploits the connection of the Delaunay triangulation and its dual, the Voronoi diagram, with a subset of k points in OPT out of n given points. Since the given points are in convex position, the Voronoi diagram of the k points will form a tree. This will allow us to use dynamic programming where we don't have cyclic dependencies in subproblems.

Initially, we choose three arbitrary points for S as the initial three optimal points, for which we consider the Delaunay triangulation and its dual Voronoi diagram. We choose the next optimal point by extending the Voronoi diagram so that this point satisfies the properties of Delaunay triangulation. In order to find this optimal point, we make all possible choices available. We repeat this until we have picked k points.

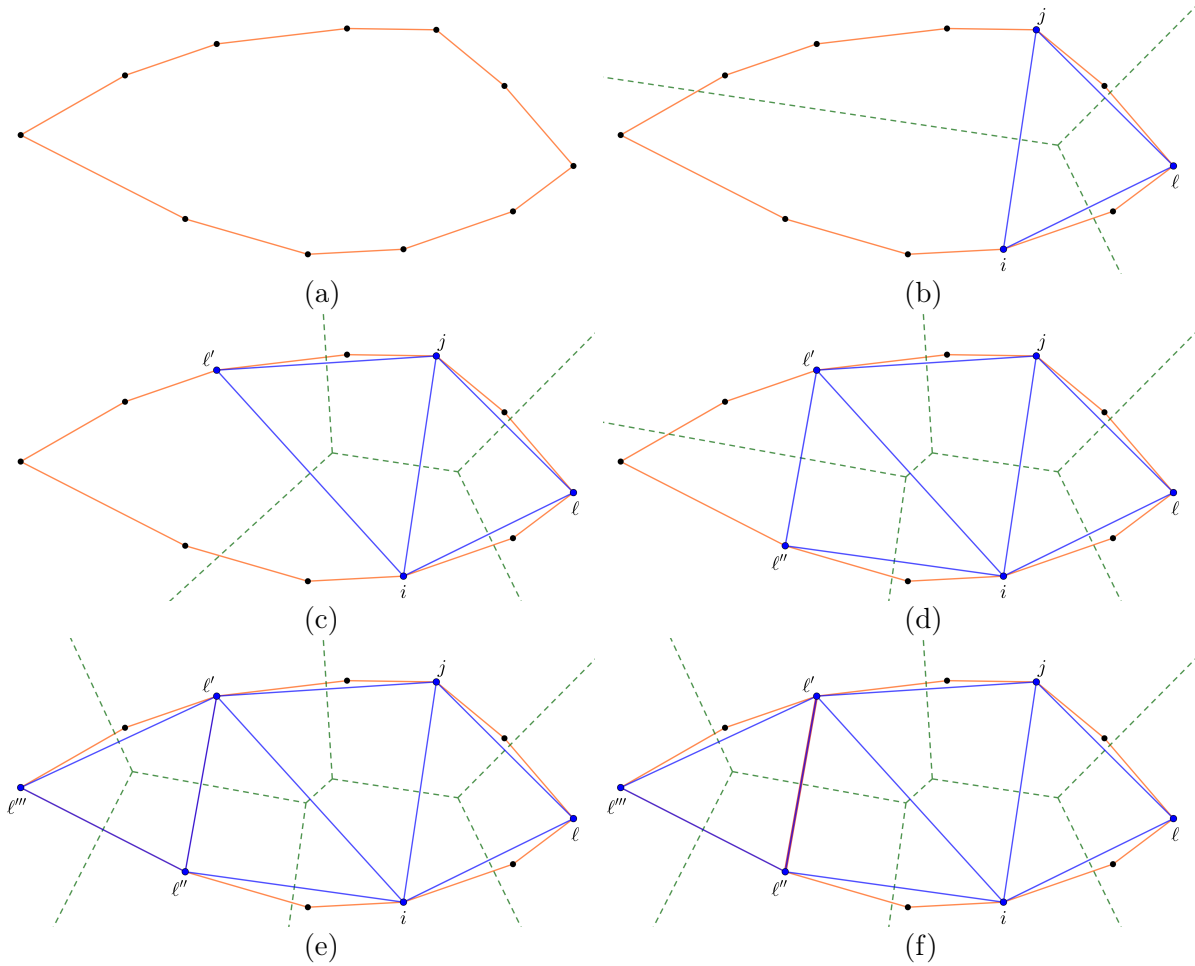
Formally, the algorithm is described as follows. Let $S' \subseteq S$ be a subset of k points that are the center vertices of disks in OPT. Consider a Delaunay triangulation $\mathcal{DT}(S')$ of these optimal k centers. Obviously, all edges of $\mathcal{DT}(S')$ must lie within the convex hull \mathcal{P} of S and also within the convex polygon Δ with S' as its vertices inscribed in \mathcal{P} . Additionally, the triangles and edges of a Delaunay triangulation of any set of points have some nice properties. As the Delaunay triangulation maximizes the minimum angle and follows the empty circle property [50], we have the following standard observation.

Observation 5.5. The shortest diagonal or edge of Δ is always a Delaunay edge of $\mathcal{DT}(S')$.

By Observation 5.5, another alternative way of viewing the DKCONP problem is to solve the problem of computing a subset $S''(\subseteq S)$ with k points such that the shortest edge of $\mathcal{DT}(S'')$ is as long as possible.

We know that the dual graph of a $\mathcal{DT}(S')$ is a Voronoi diagram $\mathcal{VD}(S')$. Since the k points that are in S' are in a convex position, $\mathcal{VD}(S')$ is a tree. For example, in Figure 5.4, the edges colored blue represent the Delaunay triangulation, while the dashed green lines form its dual, the Voronoi diagram, which is a tree. In this example, we considered a convex polygon with 11 vertices (see Figure 5.4 (a)). Using the Delaunay triangulation for $k = 6$, the OPT is constructed by selecting the rightmost three points i , j , and k , forming the initial Delaunay triangle along with its dual Voronoi diagram (see Figure 5.4 (b)). For the subsequent Delaunay triangle, a point ℓ' is chosen as part of the OPT, situated to the left of \overline{ij} and satisfying the empty circle property, with its dual Voronoi extended to ℓ' (see Figure 5.4 (c)). Similarly, the Delaunay and corresponding Voronoi diagrams are extended to points ℓ'' and ℓ''' as illustrated in Figure 5.4 (d) and Figure 5.4 (e), respectively. Finally, the optimal radius is determined by the minimum length edge in the Delaunay triangulation, represented by $\overline{\ell'\ell''}$ (see Figure 5.4 (f)). This allows us to design a dynamic programming algorithm to solve the above optimization problem as follows:

Let e_1 be an edge in $\mathcal{VD}(S')$ corresponding to the bisector of the pair $p_i, p_j \in S'$ (see Figure 5.5). Now, assume that we know the three points p_i, p_j, p_ℓ are in the optimal solution S' such that the triangle $\Delta(p_i, p_j, p_\ell)$ formed by these points is a Delaunay triangle in $\mathcal{DT}(S')$. Let the segment (p_i, p_j) be oriented from p_i to p_j . Let $\mathcal{K} \in \mathbb{Z}^+$ be the budget (i.e., the number of facility centers remaining to be selected) and $\mathcal{E}_c(p_i, p_j, p_\ell)$ be a circle circumscribing the points p_i, p_j, p_ℓ . We define a subproblem $\phi(i, j, \ell; \mathcal{K})$, which returns the length of the smallest edge or diagonal of the optimal solution S'' (to DKCONP with $k = \mathcal{K} + 3$) in which the centers p_i, p_j, p_ℓ are already

FIGURE 5.4: OPT using Delaunay triangulation when $k = 6$.

present, forming the Delaunay triangle in $\mathcal{DT}(S'')$ and \mathcal{K} is the number of remaining centers to be selected. To solve $\phi(i, j, \ell; \mathcal{K})$, we need to find the best $p_{\ell'} \in S$ that lies to the left of line through $\overrightarrow{p_i p_j}$ (this region is the left half-plane of $\overrightarrow{p_i p_j}$, denoted as $L_{\overrightarrow{p_i p_j}}^-$), outside of $\mathcal{E}_c(i, j, \ell)$ such that it partitions \mathcal{K} into two sub-parts \mathcal{K}' and $\mathcal{K} - \mathcal{K}' - 1$ optimally, where \mathcal{K}' is a positive integer. Now, recursively solve the subproblems $\phi(p_i, p_{\ell'}, p_j; \mathcal{K}')$ and $\phi(p_{\ell'}, p_j, p_i; \mathcal{K} - 1 - \mathcal{K}')$ lying to the left of $\overrightarrow{p_i p_{\ell'}}$ and to the left of $\overrightarrow{p_{\ell'} p_j}$ respectively (see Figure 5.5). These two subproblems are invoked on only those points in S that lie to the left of the line through $\overrightarrow{p_i p_{\ell'}}$ and to the left of the line through $\overrightarrow{p_{\ell'} p_j}$, and that do not lie in the interior of $\mathcal{E}_c(p_{\ell'}, p_i, p_j)$. Hence we have the following recurrence:

$$\phi(p_i, p_j, p_{\ell}; \mathcal{K}) = \max_{\substack{\mathcal{K}' \leq \mathcal{K} - 1, \\ p_{\ell'}: p_{\ell'} \notin \mathcal{E}_c(p_i, p_j, p_{\ell}), \\ p_{\ell'} \in L_{\overrightarrow{p_i p_j}}^-, p_{\ell'} \in S}} \left\{ \min \left\{ \begin{array}{l} |p_i p_{\ell'}|, |p_{\ell'} p_j|, \\ \phi(p_i, p_{\ell'}, p_j; \mathcal{K}'), \\ \phi(p_{\ell'}, p_j, p_i; \mathcal{K} - 1 - \mathcal{K}') \end{array} \right\} \right\}$$

The base cases are the following:

- $\phi(p_i, p_j, p_\ell; 1) = \max_{\substack{p_{\ell'}: p_{\ell'} \notin \mathcal{E}_c(p_i, p_j, p_\ell), \\ p_{\ell'} \in L_{\overrightarrow{p_i p_j}}^-, p_{\ell'} \in S}} \{\min\{|p_i p_{\ell'}|, |p_{\ell'} p_j|\}\}$, when we have only one facility left to select.
- $\phi(p_i, p_j, *; 1) = \max_{\substack{p_{\ell'}: p_{\ell'} \notin \mathcal{E}_c(p_i, p_j, p_\ell), \\ p_{\ell'} \in L_{\overrightarrow{p_i p_j}}^-, p_{\ell'} \in S}} \{\min\{|p_i p_{\ell'}|, |p_{\ell'} p_j|\}\}$, where $*$ indicates that there is no point p_ℓ on the right of $\overrightarrow{p_i p_j}$ (i.e., $\overrightarrow{p_i p_j}$ may be an edge of convex hull of S').
- $\phi(p_i, p_j, p_\ell; 0) = |p_i p_j|$, when all k facilities are already located.

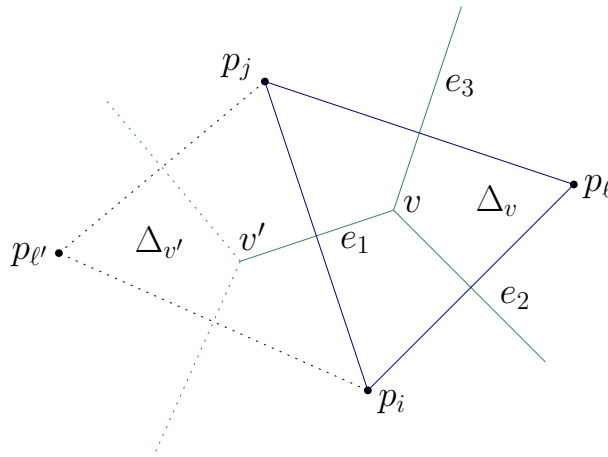


FIGURE 5.5: Extending Voronoi diagram to next Voronoi center v' by picking vertex $p_{\ell'}$.

In the above recurrence, there is no cyclic dependency between subproblems as the budget is partitioned into sub-parts for subproblems (i.e., the budget gets smaller and smaller for the subproblems). The optimal solution corresponds to a cell in the two-dimensional array of $\phi(p_i, p_j, *, k - 2)$ of the four-dimensional DP table.

Correctness of the above dynamic programming algorithm:

The correctness follows from the fact that the dual to the Delaunay triangulation \mathcal{DT}_{OPT} of an optimal solution is a tree \mathcal{VD}_{OPT} . Here, assume that we have one vertex v of \mathcal{VD}_{OPT} . Let Δ_v be the corresponding Delaunay triangle in \mathcal{DT}_{OPT} (see Figure 5.5). We know there are three edges e_1, e_2 and e_3 incident on v . We correctly determine the other endpoint v' of one of these three edges e_1, e_2 and e_3 by finding a $\Delta_{v'}$ that satisfies the empty circle property and maximizes the minimum Delaunay edge length with the best partitioning of the budget. From v' , the Voronoi diagram will be extended recursively until the budget cannot be further divided. Since we reached v' from v , at v' we have two possible ways to find the next vertex of the tree \mathcal{VD}_{OPT} . The vertex v corresponds to any triple $p_i, p_j, p_{\ell'}$ that forms a Delaunay triangle, where

p_i, p_j forms an edge of the convex hull of any feasible solution S'' . As we are checking all possible combinations of optimal Delaunay triangulations that maximize the minimum edge length of \mathcal{DT} , the above dynamic programming will return the optimal solution, and the optimal value can be obtained from one of the $O(n^2)$ cells $\phi(p_i, p_j, *, k - 2)$ in the DP table, where p_i, p_j corresponds to an edge of the convex hull of S' . Particularly, the optimal value is obtained from one of the $O(n^2)$ cells $\phi(p_i, p_j, *, k - 2)$ in the DP table. To construct S' , we will backtrack from the cell $\phi(p_i, p_j, *, k - 2)$ that has the maximum value and the final optimal value is calculated as

$$\max_{1 \leq i < j \leq n} \{\min\{|p_i p_j|, \phi[p_i, p_j, *, k - 2]\}\}.$$

Theorem 5.6. *We can solve the DKCONP problem in $O(n^4 k^2)$ time using dynamic programming.*

Proof. Note that there are $O(n^3 k)$ subproblems. In each subproblem, we have $O(n)$ choices to select $p_{\ell'}$ and $O(k)$ choices to partition \mathcal{K} . So, we spend $O(nk)$ time to combine optimal solutions to subproblems into an optimal solution to the bigger subproblem. Hence, the total running time of the dynamic programming algorithm for the DKCONP problem for any $k > 0$ is $O(n^4 k^2)$. \square

5.4 A linear time $1/2\sqrt{2}$ -approximation for $k = 3$

In this section, we propose an approximation algorithm for the DKCONP problem in $O(n)$ time for $k = 3$ if the points are given in convex position order in the first quadrant and stored in some data structure in that order. Consider a set S of n points in a general convex position. Let a, b, c, d be the four extreme points of the convex polygon \mathcal{P} formed by the given n points. Without loss of generality, let a be the extreme point with the minimum x -coordinate, b with the maximum y -coordinate, c with the maximum x -coordinate, and d with the minimum y -coordinates. Based on the position of the given n points, we have three possible situations for the positions of extreme points of \mathcal{P} : (i) all the four points a, b, c, d are distinct, (ii) two among four points coincide (i.e., $a = b, c, d$), and (iii) two pair of points coincide (i.e., $a = b, c = d$).

Let \overline{pq} be the perpendicular bisector for the line segment joining $u, v \in \{a, b, c, d\}$, where $u \neq v$. Now consider the following cases:

case 1: Let e be a point of S farthest from the line through \overline{uv} . Now, we place disks centered at u, v and e such that the radius $r_{l1} = \min\{|uv|, |ue|, |ev|\}/2$ (see Figure 5.6(a)). This is repeated for every (u, v) , where $u, v \in \{a, b, c, d\}$, $u \neq v$ and let the corresponding radii be r_{l1} for $l = 1, 2, \dots, \binom{4}{2}$.

case 2: Let $f \in S$ be the nearest point to the perpendicular bisector \overline{pq} . Now, we place disks centered at u, v and f such that the radius $r_{l2} = \min\{|uv|, |uf|, |fv|\}/2$ (see Figure 5.6(b)). This is repeated for every (u, v) , where $u, v \in \{a, b, c, d\}$, $u \neq v$ and r_{l2} is the corresponding radii for $l = 1, 2, \dots, \binom{4}{2}$.

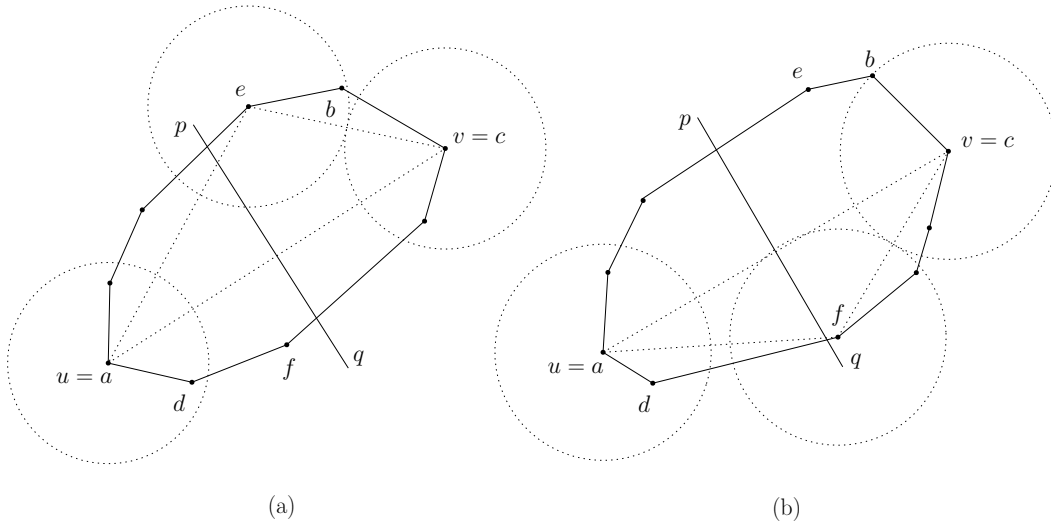


FIGURE 5.6: (a) e is the farthest point from the line segment \overline{ac} . (b) f is the nearest point to the perpendicular bisector \overline{pq} .

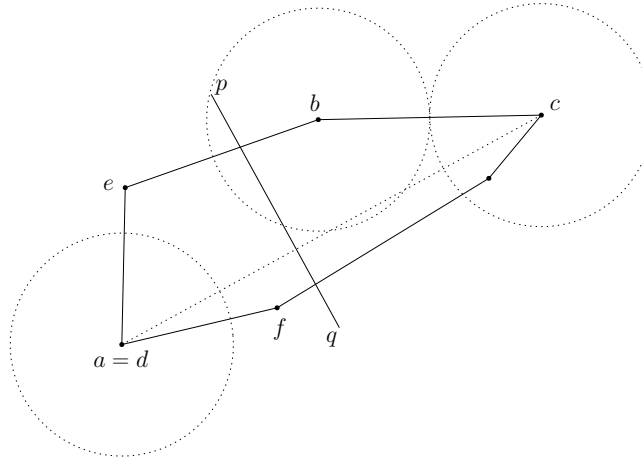


FIGURE 5.7: The disks are centered at extreme points a, b and c .

case 3: We place disks centered at three of four extreme points such that their common radius is maximized, i.e.,

$$r_1 = \max_{u \neq v \neq w \in \{a,b,c,d\}} \{\min\{|uv|, |vw|, |uw|\}\}/2.$$

Let $\{r_{l1}, r_{l2}\}_{l=1}^6 = \bigcup_{l=1}^6 \{r_{l1}, r_{l2}\}$. In the remainder of this section, we argue that the $\max(\{r_1\} \cup \{r_{l1}, r_{l2}\}_{l=1}^6)$ will be an $1/2\sqrt{2}$ -approximate value for the radius in the optimal packing when $k = 3$, i.e., $\max(\{r_1\} \cup \{r_{l1}, r_{l2}\}_{l=1}^6) \geq r_{max}/2\sqrt{2}$.

The extreme points of a convex polygon can be found in $O(\log n)$ time when the vertices of the polygon are given in a clockwise or counter-clockwise order [58]. This $O(\log n)$ -time search algorithm is a minor variant of binary search, as described in Chapter 7 of [58]. It uses the notion of directed edges of the polygon to determine how to halve the search interval based on the given order (clockwise or counter-clockwise) of the points. Consequently, this approach enables us to

do a binary search over the set S . Therefore, we can determine the extreme points a , b , c and d of the set S in $O(\log n)$ time.

Now, let d_l be a diagonal formed by any two of these extreme points. Now, we can observe that the boundary $\partial\mathcal{P}$ can be split into two convex polygonal chains above and below d_l , respectively. Hence, due to the convexity of \mathcal{P} , we can find the farthest point from the line through d_l with a similar adaption of the binary search [58] in $O(\log n)$ time. We can find a perpendicular bisector of the line through d_l in constant time and the points of the convex polygon closest to the perpendicular bisector in $O(\log n)$ time with a similar approach (because here, unimodality property holds for the distances between bisector line and the vertices). In total, there are at most four such points. Therefore, the running time required is $O(\log n)$ in all the aforementioned cases when the vertices of the convex polygon are stored in a data structure in clockwise/counter-clockwise order. However, the overall running time of the approximation algorithm is $O(n)$ since we read the input array storing the set S , i.e., the vertices of the convex polygon in clockwise/counter-clockwise order. As a result, we have the following theorem.

Theorem 5.7. *We have a linear time $1/2\sqrt{2}$ -approximation algorithm for the DKCONP problem when $k = 3$.*

Proof. The running time of the above algorithm is clearly $O(\log n)$, as every step of the algorithm takes either $O(1)$ or $O(\log n)$ time. Reading the input points S into the memory in a clockwise/counter-clockwise order will take $O(n)$ time. Hence, the total time taken by the algorithm is $O(O(n) + \log n) = O(n)$. Now, we argue about its approximation factor. Let r_{max} be the optimal radius of the disks in the optimal solution. Then r_{max} is at most half of the diameter of \mathcal{P} , i.e., $r_{max} \leq \frac{\mathcal{D}(\mathcal{P})}{2}$. Let the convex polygon \mathcal{P} be enclosed inside an axis-parallel rectangle \mathcal{R} such that the extreme points of \mathcal{P} lie on the boundary of \mathcal{R} (see Figure 5.8).

case 1: When $r_1 > \max(\{r_{l1}, r_{l2}\}_{l=1}^6 \setminus \{r_1\})$. Without loss of generality assume that $r_1 \geq |ad|/2$ and that $|a'd|/2 = \max(\{r_{l1}, r_{l2}\}_{l=1}^6 \setminus \{r_1\})$ and thus $|a'd| \leq |ad|$

Let $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ be top-left, top-right, bottom-right, bottom-left corner vertices of \mathcal{R} . If we first assume that $|ad| < \frac{|\alpha_3\alpha_4|}{2}$, then since $|a'd| \leq |ad|$ we have $r_1 = \frac{|ad|}{2} = r_{max}$. Otherwise we can consider the case (see figure 5.8(a)) where we have $|ad| \geq \frac{|\alpha_3\alpha_4|}{2}$. Let θ_1 be the angle at α_3 defined by $\overline{\alpha_1\alpha_3}$ and $\overline{\alpha_4\alpha_3}$. Observe that $\theta_1 \leq 45^\circ$. Therefore,

$$\cos(\theta_1) \geq \frac{1}{\sqrt{2}} \implies |\alpha_4\alpha_3| \geq \frac{1}{\sqrt{2}}|\alpha_1\alpha_3| \quad (5.1)$$

Since $r_{max} \leq \frac{|\alpha_1\alpha_3|}{2}$ and $|ad| \geq \frac{|\alpha_3\alpha_4|}{2}$ where $r_1 = \frac{|ad|}{2}$, then we have,

$$r_1 \geq \frac{|\alpha_3\alpha_4|}{4} \geq \frac{1}{4\sqrt{2}}|\alpha_1\alpha_3| \geq \frac{1}{2\sqrt{2}}r_{max}$$

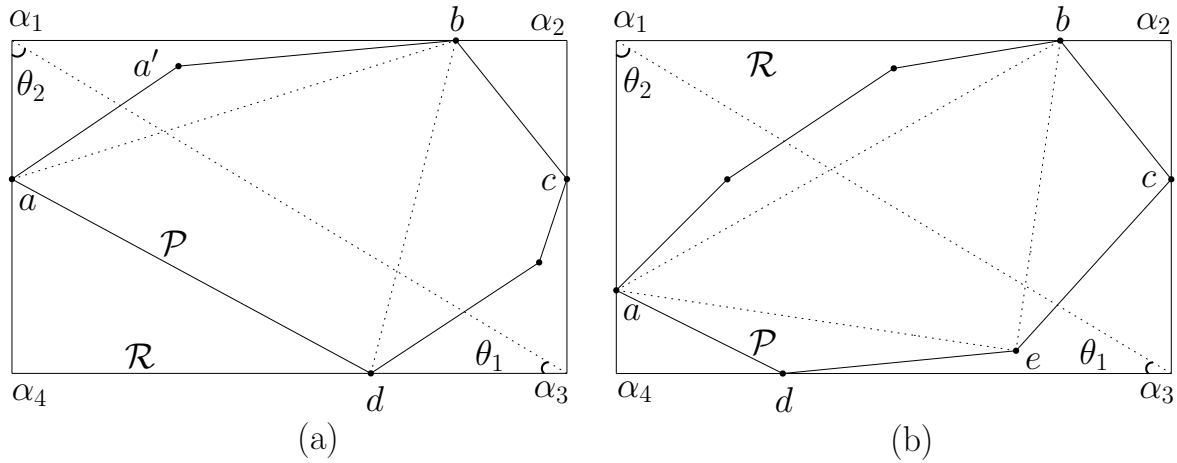


FIGURE 5.8: (a) When $r_1 > \max(\{r_{l_1}, r_{l_2}\}_{l=1}^6 \setminus \{r_1\})$. (b) When $r_{l^*m} = \max\{r_{l_1}, r_{l_2}\}_{l=1}^6$ and $r_{l^*m} > r_1$.

case 2: When $r_{l^*m} = \max\{r_{l_1}, r_{l_2}\}_{l=1}^6$ and $r_{l^*m} > r_1$, where $l^* \in \{1, 2, \dots, 6\}$, and $m = 1$ or $m = 2$.

Let e be the farthest point from the line through the segment \overline{ab} . Without loss of generality assume that r_{l^*m} corresponds to $\{a, b, e\}$. In figure 5.8(b) we can clearly observe that $|ae| \geq \frac{|\alpha_3\alpha_4|}{2}$, because otherwise it would imply that any vertex f above \overline{ab} along with c and e corresponds to r_{l^*m} contradicting the choice of r_{l^*m} . Therefore, we have the same series of inequalities (6.1). Since $r_{max} \leq \frac{|\alpha_1\alpha_3|}{2}$ and $|ae| \geq \frac{|\alpha_3\alpha_4|}{2}$ where $r_{l^*m} = \frac{|ae|}{2}$, then we have,

$$r_{l^*m} \geq \frac{|\alpha_3\alpha_4|}{4} \geq \frac{1}{4\sqrt{2}}|\alpha_1\alpha_3| \geq \frac{1}{2\sqrt{2}}r_{max}$$

Hence, the algorithm chooses two disk centers at the endpoints of d_l while the third disk is placed based on the above-mentioned three cases, which gives the radius that is at least $\frac{r_{max}}{2\sqrt{2}}$. \square

5.5 Conclusion

In this chapter, we studied the k -dispersion problem on convex set of points in the plane and proposed: (i) an exact fixed-parameter algorithm with runtime $O(2^k n^2 \log^2 n)$, (ii) an exact polynomial time algorithm with runtime $O(n^4 k^2)$ for any $k > 0$. For practical purposes, one may prefer (i) for small values of k and (ii) for large values of k as the polynomial dependency on n is smaller for the fixed-parameter algorithm than the polynomial time algorithm. Finally, for $k = 3$ we proposed a linear time $\frac{1}{2\sqrt{2}}$ -approximation algorithm.

The results in this chapter are published in [75] and submitted for publication [74].

Chapter 6

Edge-Vertex Domination in UDG

Dominating set is a classical and popular combinatorial optimization problem in algorithmic graph theory. A *dominating set* in an undirected graph $G = (V, E)$ is a subset $V^d \subseteq V$ such that every vertex $v \in V$ is either in V^d or adjacent to a vertex in V^d . A graph can have many dominating sets. We refer to a dominating set (V^d) with the minimum cardinality as the *minimum dominating set* (MDS). The *domination number* is the cardinality of the MDS, and it is denoted by $\gamma(G)$.

Finding a minimum dominating set in a UDG has applications in facility location, where the goal is to minimize the number of facilities (e.g., gateways or sink nodes) required to cover all points of interest (e.g., network nodes). For instance, determining the minimum number of wireless access points needed to ensure complete coverage of all potential users can be solved by formulating it as the minimum dominating set problem in a UDG representing the area in which the users are located.

In this chapter, we consider a variant of the dominating set problem known as *edge-vertex dominating set* (EVDS) problem on unit disk graphs.

Definition 6.1 (Edge-vertex dominating set (EVDS)). Given an undirected graph $G = (V, E)$, a vertex $v \in V$ is *ev (edge-vertex)-dominated* by an edge $e \in E$ if v is incident to e (i.e., an endpoint of e) or if v is incident to an adjacent edge of e . A set $S^{ev} \subseteq E$ is an *edge-vertex dominating (EVD) set* (referred to as *ev-dominating set*) of G , if for each vertex $v \in V$, there exists an edge $e \in S^{ev}$ that e *ev-dominates* v . The minimum cardinality of an *ev-dominating set* is the *ev-domination number*, denoted by $\gamma_{ev}(G)$. The EVDS problem is to find $\gamma_{ev}(G)$.

Similar to other dominating set variants in UDG in the literature, the EVDS problem is also a combinatorial optimization problem, hence selecting the minimum number of edges to *ev-dominate* all the vertices is challenging due to the exponential number of possible subsets of E . However, by exploiting some geometric properties of unit disk graphs connected to EVDS, we

attempt to give efficient approximation algorithms for the problem. In this chapter, we first show that the decision version of the EVDS problem is **NP-complete** in UDGs. We also prove that this problem on UDG admits a polynomial time approximation scheme (PTAS). Finally, we present a simple 5-factor linear-time approximation algorithm.

6.1 Hardness results

In this section, we show that the decision version of the EVDS is **NP-complete**, as stated below. We describe a polynomial time reduction from the vertex cover problem, which is known to be **NP-complete** in planar graphs with maximum degree 3 [32], to the EVDS problem on UDG. For the reduction, we first consider a planar graph with a maximum degree of 3. We then embed the planar graph onto a grid in the Euclidean plane, placing each vertex of the planar graph at integer coordinates and embedding the edges as a series of line segments with at most two bends. Next, we divide each edge into smaller edges based on specific distance constraints, as discussed below, to ensure that there is a vertex cover of size at most k for the planar graph if and only if there exists an EVDS of size at most $k + l$ for the embedded graph, where k is a positive integer and l is the number of line segments in the embedding.

The EVDS problem on UDGs (EVDS-UDG)

Instance: A UDG $G = (V, E)$ and a positive integer k .

Question: Does there exist an edge-vertex dominating set S^{ev} of G such that $|S^{ev}| \leq k$.

Lemma 6.2. ([78]) *An embedding of a planar graph $G = (V, E)$ with maximum degree 4 in the plane is possible such that this embedding uses only $O(|V|^2)$ area and its vertices are at integer coordinates, and its edges are drawn so that they are along the grid line segments of the form $x = i$ or $y = j$, for some i and j , where $i, j \in \mathbb{Z}^+$.*

Biedl and Kant [11] gave a linear time algorithm that produces this kind of embedding (embedding in Lemma 6.2) in which each edge contains at most two bends (see Figure 6.1, where each edge of the planar graph G have at most two bends in its embedding and $l = 15$).

Corollary 6.3. ([42]) *An embedding of a planar graph $G = (V, E)$ with $|V| \geq 3$ and maximum degree 3 in the plane can be constructed in polynomial time, where the embedding is such that the vertices of G are at $(4i, 4j)$ and the edges of G are drawn as a sequence of consecutive line segments along the lines $x = 4i$ or $y = 4j$, for some i and j .*

Lemma 6.4. *Let $G = (V, E)$ be an instance of the vertex cover problem on a planar graph with at least two edges and a maximum degree of three. An instance $G' = (V', E')$ of EVDS-UDG can be constructed from G in polynomial time.*

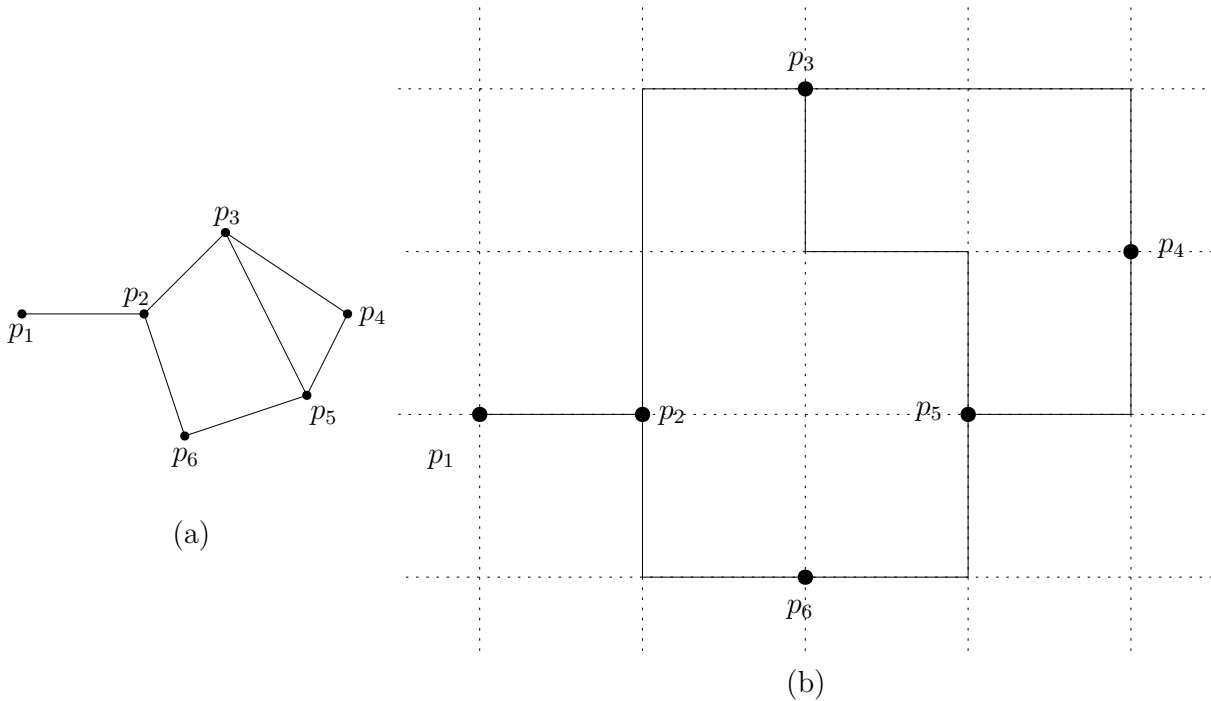


FIGURE 6.1: (a) A planar graph G with max degree 3, and (b) The embedding of G on a grid.

Proof. The construction of $G' = (V', E')$ from $G = (V, E)$ is as follows: First, using one of the algorithms discussed in [38, 40] we embed the graph $G = (V, E)$ into a grid of size $4n \times 4n$ such that each edge of E is composed of a sequence of horizontal or vertical line segment(s), each of whose length is four units long. The points $\{p_1, p_2, \dots, p_n\}$ are referred to as the *node points* in the embedding with respect to the vertex set of G (see Figure 6.1(a), Figure 6.1(b) and the corresponding UDG G' in Figure 6.2). In the embedded graph, for each edge of length greater than four units, we add a joint point to join two line segments in the embedding other than the node points. Name these points as the *joint points* (see empty circles in Figure 6.2). Then for each line segment with *joint points* as both of its endpoints in the embedding, we add three extra points such that each of these extra points is at a distance of 1 unit from its neighbor extra point(s) placed on the same segment, at least 1 unit from the corresponding joint points. Similarly, for each line segment with a *node point* as its endpoint, we add four extra points each at a distance of 0.8 unit from its neighbor extra point(s), also from the endpoints of the segment on which we are placing the extra points. Name these extra points (from both the above cases) as the *added points* (see filled square points in Figure 6.2).

Let A be the set of added points and J be the set of joint points. We construct a UDG $G' = (V', E')$ where the vertex set $V' = V \cup A \cup J$, and there is an edge between two vertices of V' if and only if the distance between them is at most 1 unit (see Figure 6.2). If l is the total number of line segments in the embedding, then $|A| \leq 4l$ and $|J| \leq (l - |E|)$. It follows from Lemma 6.2 that l is at most $O(n^2)$. Clearly, the graph defined by the intersection of unit disks

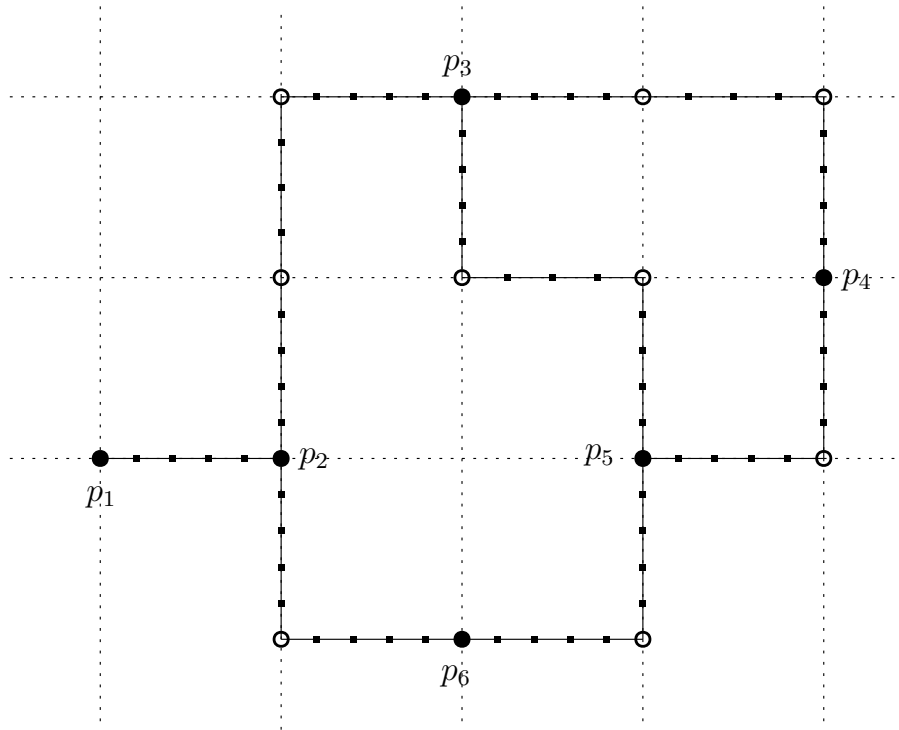


FIGURE 6.2: A unit disk graph construction from embedding.

centered at points in V' is a unit disk graph. Since both the sets $|V'|$ and $|E'|$ are bounded by $O(n^2)$, we can construct G' from G in polynomial time. \square

Lemma 6.5. $EVDS\text{-}UDG \in \text{NP}$.

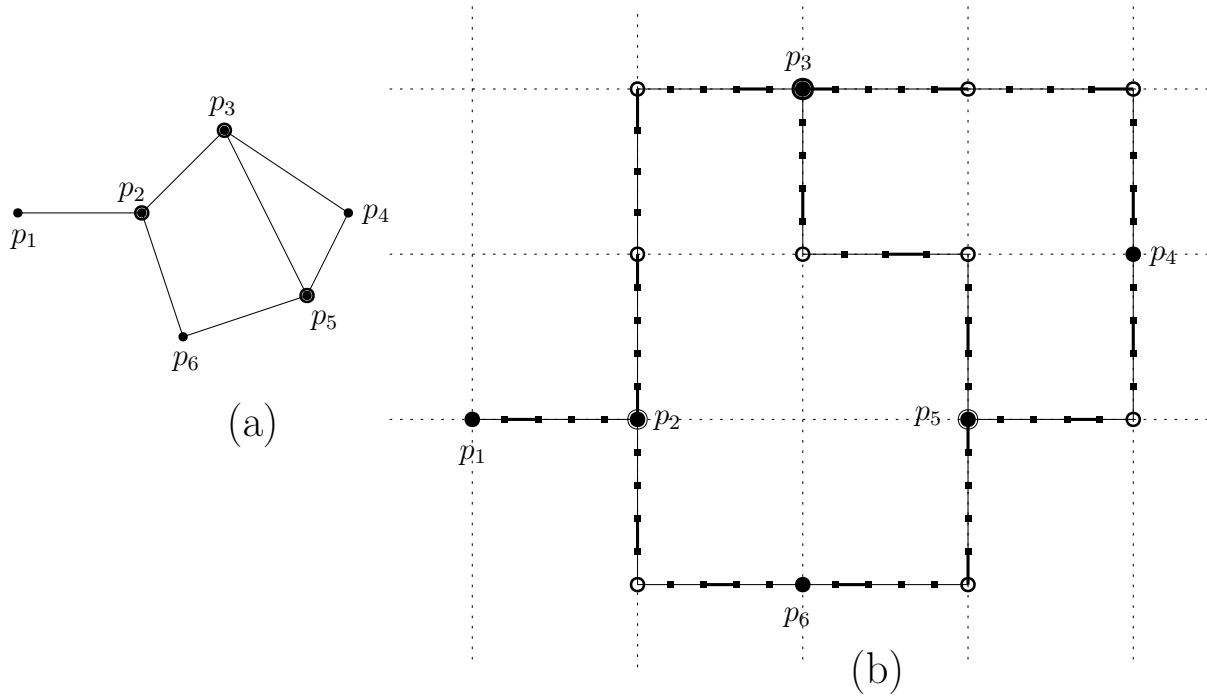
Proof. Given a subset $S^{ev} \subseteq E$ and a positive integer k , we can verify that S^{ev} is an edge-vertex dominating set of size at most k in polynomial time by checking whether each vertex $v \in V$ is ev -dominated by an edge $e \in S^{ev}$ all in $O(|V'||E'|)$ time. \square

We prove the NP-hardness of the EVDS-UDG problem by reducing the decision version of the *vertex cover* problem on a planar graph with maximum degree 3 to the EVDS-UDG problem. Let $G = (V, E)$ be a planar graph with a maximum degree of 3. Then from Lemma 6.4, we can construct an instance $G' = (V', E')$ of EVDS-UDG in polynomial time.

Lemma 6.6. G has a vertex cover of size at most k if and only if G' has an edge-vertex dominating set of size at most $k + l$.

Proof. Let $D \subseteq V$ be a vertex cover of G of cardinality at most k . Let D' be the set of vertices that are the *node points* of G' corresponding to the vertices in D .

Necessary: Now, for every vertex v in D' , choose any one edge of G' that is incident to v where the path does not lead to a pendant vertex (i.e., a vertex with degree one) in G . Represent these

FIGURE 6.3: (a) Vertex cover of G . (b) EVDS of UDG.

chosen edges as the set N' . Since $|D'| \leq k$, the cardinality of N' is at most k . We can see that for any edge $(p_i, p_j) \in E$ of G , we have a simple path in G' , consisting of at least one line segment and only added points and joint points between p_i and p_j . Let us introduce the notation $p_i \rightsquigarrow p_j$ to denote this path for any pair of node points $(p_i, p_j) \in E$, where the node point p_i is a vertex in D' . Next, traverse every such path exactly once starting at a vertex $v \in D'$, and initially choose the fourth edge from $(v, u) \in N'$ (not counting (v, u)) and then on every fourth edge until reaching p_j . We repeat this for every vertex $v \in D'$ except for the paths that will lead to pendant vertices of G , which will ensure that every path $p_i \rightsquigarrow p_j$ is traversed exactly once because D is a vertex cover. Similarly, for every path $p_i \rightsquigarrow p_j$, where $p_i \in D$ and p_j is a pendent vertex, choose the second edge of G' from p_j and continue selecting every fourth edge until reaching p_i . Let N'' be the set of all these chosen edges. Observe that the edges in $N' \cup N''$ are chosen such that there is at least one edge and at most two edges contained in $N' \cup N''$ for each of l segments (see the darkened edges in Figure 6.3(b)). Moreover, the way edges in $N' \cup N''$ are chosen ensures that every two added or joint or node points between any two consecutive of these edges are ev -dominated by them. The cardinality of N'' is l since each line segment consists of at most four added points in the embedding. Therefore, $N' \cup N''$ is an ev -dominating set for G' and $(|N'| + |N''|) \leq k + l$.

Sufficiency: To prove the sufficiency, consider any edge-vertex dominating set $S^{ev} \subseteq E'$ for G' , of size at most $k + l$. We know that for any node point $p_i \in V'$, the degree of p_i is at most 3 in both G and G' . Let $p_i \rightsquigarrow p_{j_t}$ ($t = 1, 2, 3$) be the three paths in G' as defined above, i.e., all the other vertices through which the path $p_i \rightsquigarrow p_{j_t}$ traverses are only the joint points and added points.

Let $\zeta(p_i \rightsquigarrow p_{j_t})$ be the subset of edges of E' that appear in the path $p_i \rightsquigarrow p_{j_t}$. For any node point p_i in G' , let $\ell_t = |\zeta(p_i \rightsquigarrow p_{j_t}) \cap S^{ev}|$ be the number of edges of that path contained in the EVDS S^{ev} . Let j_t be the number of line segments that constituted the path $p_i \rightsquigarrow p_{j_t}$ in the embedding. Observe that ℓ_t is equal to j_t or $j_t + 1$ due to the construction of G' from the embedding of G . Now, identify a node point p_i in G' such that the degree of p_i is at least 2 and exactly one path $p_i \rightsquigarrow p_{j_1}$ has its ℓ_1 equal to $j_1 + 1$ and the remaining paths $p_i \rightsquigarrow p_{j_2}$ (and $p_i \rightsquigarrow p_{j_3}$) have their counts $\ell_2 = j_2$ (and $\ell_3 = j_3$). Pick this node point p_i into a vertex cover D . Remove the part of G' induced by these paths $p_i \rightsquigarrow p_{j_1}$, $p_i \rightsquigarrow p_{j_2}$, and $p_i \rightsquigarrow p_{j_3}$ (however, retain the node points p_{j_1} , p_{j_2} , and p_{j_3} in the remaining G'). This will guarantee that the edges of G corresponding to these paths are covered by the vertex $p_i \in D$. Repeat this procedure on the remaining G' . To start with, there must exist at least one such p_i ; otherwise, the sum $\sum_{p_i \rightsquigarrow p_{j_t}} \ell_t$ over all paths in G' would exceed $k + l$, a contradiction. Hence, D is a vertex cover for G and $|D| \leq k$.

The construction of S^{ev} from D and vice versa both take polynomial time. Thus the lemma follows. \square

Theorem 6.7. *The EVDS-UDG problem is NP-complete.*

Proof. Follows from Lemmas 6.5 and 6.6. \square

6.2 Polynomial time approximation scheme

In this section, we propose a PTAS for the EVDS set problem in a UDG. It is based on the concept of p -separated collection of subsets, which was introduced by Nieberg and Hurink [57]. The subgraphs generated by these p -separated collections will divide the graph into smaller parts such that these subgraphs will have pairwise disjoint edge vertex dominating sets. For the EVDS problem, the value of p will be set to 4 in order to satisfy this condition. This division makes it easier to find the local EVDS of the subgraphs by determining the maximal matching of each subgraph. Here, we present various properties that allow us to bound the cardinalities corresponding to the global optimal solution. Finally, we also show that the union of local EVDS in the subgraphs can be extended to become an approximate EVDS to the original graph. Then, in the following subsection, we discuss an efficient way of finding these subgraphs. This concept of p -separated collection of subsets was used by many other authors to develop PTAS (for e.g., the Roman dominating set [68], minimum Liar's dominating set [41], vertex-edge dominating set [42]). However, we adopted the concept here quite differently from these as we have to select edges to dominate vertices in the EVDS problem.

Let $G = (V, E)$ be a UDG. Let $h(e_1, e_2)$ denote the minimum number of edges in a simple path between the farthest pair of endpoints among the four endpoints of the edges e_1 and e_2 . Consider

any two subsets $E_1 \subseteq E$ and $E_2 \subseteq E$, $h(E_1, E_2)$ is defined as the minimum number of edges between any two edges $e_1 \in E_1$ and $e_2 \in E_2$. We use $EVD(A)$ to denote an ev-dominating set and $EVD_{opt}(A)$ to denote the optimal ev-dominating set of the edge-induced subgraph corresponding to $A(\subseteq E)$ (i.e., the subgraph induced by the set of edges $A(\subseteq E)$ and the endpoints of edges in A).

Definition 6.8. Let S be a set of k pairwise disjoint non-empty subsets of E , i.e., $S_i \subset E$ for $i = 1, 2, \dots, k$. If $h(S_i, S_j) \geq m$, for $1 \leq i, j \leq k$ and $i \neq j$, then S is called as the p -separable collection of subsets of E (see Figure 6.4 for $m = 4$).

Lemma 6.9. In a graph $G = (V, E)$, if $S = \{S_1, S_2, \dots, S_k\}$ is a 4-separated collection of k subsets of E , then

$$\sum_{i=1}^k |EVD_{opt}(S_i)| \leq |EVD_{opt}(E)|.$$

Proof. Let A_i be the set of edges that are adjacent to edges of S_i for each $i = 1, 2, \dots, k$ and R_i the set of edges such that $R_i = S_i \cup A_i$. The edges in sets R_1, R_2, \dots, R_k are pairwise disjoint, since the set S is a 4-separated collection of subsets of edges i.e., $(R_i \cap R_j) = \emptyset$, where $i \neq j$. Hence, the edges of $EVD_{opt}(E) \cap R_i$ will ev-dominate every vertex in S_i , since $EVD_{opt}(E)$ will ev-dominate every vertex $v \in V$. On the other hand, also $EVD_{opt}(S_i) \subset R_i$ ev-dominates every vertex of S_i , with a minimum number of edges of G . This implies that $|EVD_{opt}(S_i)| \leq |EVD_{opt}(E) \cap R_i|$. For all k subsets of edges in the 4-separated collection S , we get

$$\sum_{i=1}^k |EVD_{opt}(S_i)| \leq \sum_{i=1}^k |(EVD_{opt}(E) \cap R_i)| \leq |EVD_{opt}(E)|.$$

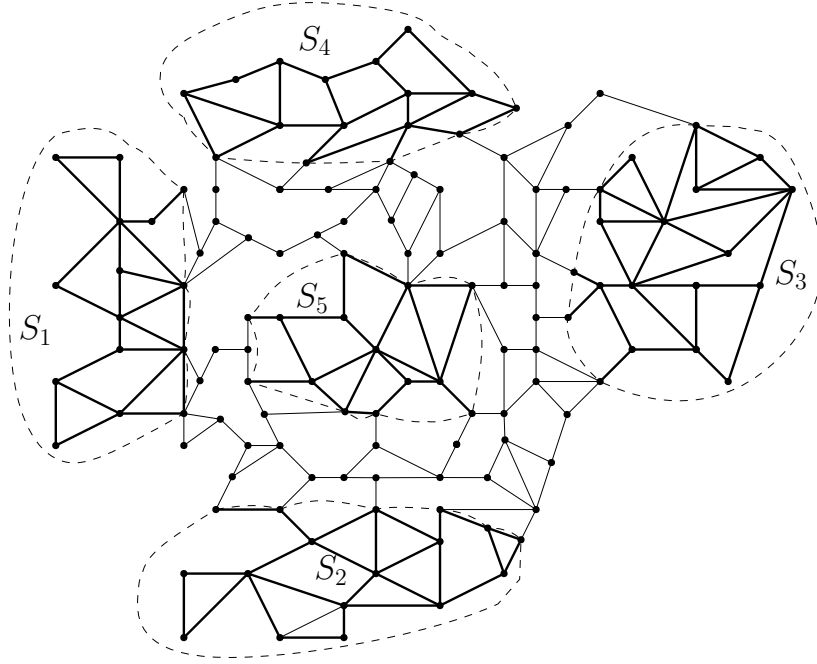
□

The above Lemma 6.9 states that a 4-separated collection of subsets of edges S will give a lower bound on the cardinality of an EVDS. Hence, we can get an approximation for the EVDS in G , if we are able to enlarge S_i to subsets $Q_i \subset E$, in such a way that EVDS of expansions are bounded locally and dominate every $v \in V$ globally.

Lemma 6.10. In a graph $G = (V, E)$, let $S = \{S_1, S_2, \dots, S_k\}$ be a 4-separated collection of subsets of edges and $Q = \{Q_1, Q_2, \dots, Q_k\}$ be a collection of subsets of E with $S_i \subseteq Q_i$ for every $i = 1, 2, \dots, k$. If there is a $\rho \geq 1$ such that

$$|EVD_{opt}(Q_i)| \leq \rho |EVD_{opt}(S_i)|$$

holds for every $i = 1, 2, \dots, k$, and if $\bigcup_{i=1}^k EVD_{opt}(Q_i)$ is an edge-vertex dominating set of G , then $\sum_{i=1}^k |EVD_{opt}(Q_i)|$ is a ρ -approximation of minimum EVDS set of G .

FIGURE 6.4: 4-Separable collection of edge sets $S = \{S_1, S_2, S_3, S_4, S_5\}$.

Proof. From Lemma 6.9 we have,

$$\sum_{i=1}^k |EVD_{opt}(S_i)| \leq |EVD_{opt}(E)|.$$

Hence, $\sum_{i=1}^k |EVD_{opt}(Q_i)| \leq \rho \sum_{i=1}^k |EVD_{opt}(S_i)| \leq \rho |EVD_{opt}(E)|$.

□

In the following section, we discuss a procedure to construct the subsets $Q_i \subset E$, that contains a 4-separated collection of edges $S_i \subset Q_i$, in such a way that a local $(1 + \epsilon)$ -approximation can be guaranteed. The union of the respective local EVDS will ev-dominate the entire vertex set of G , which results in a global $(1 + \epsilon)$ -approximation for the EVDS problem.

6.2.1 Subset Construction

Here, we discuss the construction of the 4-separated collection of subsets of edges, $S = \{S_1, S_2, \dots, S_k\}$ and the respective enlarged subsets $Q = \{Q_1, Q_2, \dots, Q_k\}$ of E such that $S_i \subseteq Q_i$ for every $i = 1, 2, \dots, k$. The basic idea of the algorithm is as follows. We start with an arbitrary edge $e \in E$ and consider the r -th edge neighborhood of e , for $r = 0, 1, 2, \dots$, with $N_e^0[e] = e$. We compute the EVDS for these edge neighborhoods until the following condition holds

$$|EVD(N_e^{r+4}[e])| > \rho |EVD(N_e^r[e])| \quad (6.1)$$

Let r_1 be the smallest r that violates the above inequality (6.1). Let $S_1 = N_e^{r_1}[e]$, $Q_1 = N_e^{r_1+4}[e]$. Then iteratively, let $S_i = N_e^{r_i}[e]$, $Q_i = N_e^{r_i+4}[e]$, $E_{i+1} = E_i \setminus (N_e^{r_i+4}(e))$ for $i = 1, 2, \dots, k$, where $E_1 = E$ and k is such that $E_{k+1} = \emptyset$. We follow this procedure iteratively for each graph induced by E_{i+1} and until $E_{i+1} = \emptyset$, finally returning the sets $S = \{S_1, S_2, \dots, S_k\}$ and $Q = \{Q_1, Q_2, \dots, Q_k\}$, where r_2, r_3, \dots, r_k are the smallest values of r violating inequality (6.1), corresponding to the 2nd, 3rd, \dots , k th iteration of the above edge-neighborhood growing procedure.

We find the edge-vertex dominating set of the r -edge neighborhood $EVD(N_e^r[e])$ of an edge e , with respect to the graph G as follows. Find a maximal matching M for the graph induced by the edges of $N_e^r[e]$. We can observe that the edges in M form an edge-vertex dominating set for the graph induced by $N_e^r[e]$. Hence, as the following lemma says, $EVD(N_e^r[e]) = M$.

Lemma 6.11. *A maximal matching M of the graph $G' = (V', E')$ induced by the edges in $N_e^r[e]$, is an EVDS of $N_e^r[e]$.*

Proof. For the contradiction, assume that M is not an EVDS of the graph $G' = (V', E')$ induced by the edges in $N_e^r[e]$. It means that there exists a vertex $v \in V'$ which is incident to an edge $e' \in E'$ such that $N_e[e'] \cap M = \emptyset$. It contradicts that M is a maximal matching in G' as the set $M \cup \{e'\}$ is a matching in G' . Thus, the lemma follows. \square

Lemma 6.12. *If $G' = (V', E')$ is a UDG induced by the edges in $N_e^r[e]$ and M is the maximal matching of G' then $|EVD(N_e^r[e])| \leq O(r^2)$.*

Proof. First, we find a maximal matching M , before finding the EVDS in $G' = (V', E')$ which is induced by the edges of $N_e^r[e]$. The number of edges in M of G' is bounded by the number of unit disks that are packed in a disk of radius $r + 2$ and centered at the middle of the edge e . Hence, $|M| \leq (r + 2)^2$ and the cardinality of $EVD(N_e^r[e])$ is bounded by $|M|$ (see Lemma 6.11). Therefore, we have

$$|EVD(N_e^r[e])| \leq |M| \leq (r + 2)^2 \leq O(r^2).$$

\square

Theorem 6.13. *There exists an r_1 which violates the following inequality.*

$$|EVD(N_e^{r_1+4}[e])| > \rho |EVD(N_e^{r_1}[e])|$$

where $\rho = 1 + \epsilon$ and r_1 is bounded by $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.

Proof. On contrary, without loss of generality for r_1 , assume that there exists an edge $e \in E$ such that

$$|EVD(N_e^{r_1+4}[e])| > \rho |EVD(N_e^{r_1}[e])|$$

for all $r \geq r_1$. Then, from Lemma 6.12, we have

$$(r + 6)^2 \geq |EVD(N_e^{r+4}[e])|.$$

Hence, when r is even we have,

$$(r + 6)^2 \geq |EVD(N_e^{r+4}[e])| > \rho |EVD(N_e^r[e])| > \dots > \rho^{\frac{r}{2}} |EVD(N_e^2[e])| \geq \rho^{\frac{r}{2}} \quad (6.2)$$

and when r is odd, we have,

$$(r + 6)^2 \geq |EVD(N_e^{r+4}[e])| > \rho |EVD(N_e^r[e])| > \dots > \rho^{\frac{r-1}{2}} |EVD(N_e^1[e])| \geq \rho^{\frac{r-1}{2}} \quad (6.3)$$

Now, we can observe that in both the inequalities (6.2), (6.3) on the left-hand side we have a polynomial in r which is at least the right-hand side value which is exponential in r , it is a contradiction. Therefore, for all $r \geq r_1$ the inequality (6.2) cannot hold, hence there exists such r_1 . Ultimately, r_1 depends only on ρ , not on the size of the edge-induced subgraph by $N_e^{r+4}[e]$. As in [57], we can argue that r_1 is bounded by $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ where $\rho = 1 + \epsilon$. \square

Lemma 6.14. *Given an $\epsilon > 0$, for an edge $e \in E$, $EVD_{opt}(Q_i)$ can be computed in polynomial time.*

Proof. From the way of construction of Q_i , we can see that $Q_i \subseteq N_e^{r+4}[e]$. The cardinality of EVDS of $N_e^{r+4}[e]$ is bounded by $O(r^2)$ (see Lemma 6.12), where r is bounded by $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ (see Theorem 6.13). Hence, we need at most $O(n^{r^2})$ possible combinations of $O(r^2)$ -tuples of vertex points to check whether the selected tuple is an EVDS of Q_i . \square

Lemma 6.15. $\bigcup_{i=1}^k EVD(Q_i)$ is an edge-vertex dominating set in $G = (V, E)$.

Proof. It follows from the construction of the collection of subsets of edges $\{Q_1, Q_2, \dots, Q_k\}$ that each edge that is incident to a vertex $v \in V$ belongs to a specific subset Q_i and $EVD(Q_i)$ is an EVDS of the graph induced by the edges of Q_i . Therefore, every vertex $v \in V$ is incident to at least one edge e such that at least one edge of $N_e[e]$ is in $\bigcup_{i=1}^k EVD(Q_i)$. \square

Corollary 6.16. $\bigcup_{i=1}^k EVD_{opt}(Q_i)$ is an edge-vertex dominating set in $G = (V, E)$, for the collection of subsets of edges $Q = \{Q_1, Q_2, \dots, Q_k\}$.

Theorem 6.17. *For a given unit disk graph and an $\epsilon > 0$, there exists a PTAS (an $(1 + \epsilon)$ -approximation) algorithm for the EVDS problem with running time $n^{O(c^2)}$, where $c = (\frac{1}{\epsilon} \log \frac{1}{\epsilon})$.*

Proof. Follows from Corollary 6.16 and Lemma 6.14. \square

6.3 5-Factor approximation algorithm

In this section, we present a 5-factor approximation algorithm for the EVDS problem on UDG. Let \mathcal{S} be a set of n points given in the Euclidean plane. We join two of these points with an edge if the distance between those two points is less than or equal to 1 unit. Let E be the set of such edges with cardinality m and V be the set of vertices corresponding to points in \mathcal{S} . The graph induced by V and E will form a UDG since the distance between any two end-points of $e \in E$ is at most 1. Assume that such a UDG has no isolated vertex, otherwise, EVDS does not exist. To present an approximation algorithm, we consider an axis-parallel rectangular region \mathcal{R} that contains UDG. We then partition the region \mathcal{R} into grid cells by a tessellation with regular hexagons, where each hexagonal cell is of side length $\frac{1}{2}$. Hence, the maximum distance between any two points inside a cell is at most 1. Assume that no point in V lies on the boundary of any hexagon in the partition.

Lemma 6.18. *Any edge e with its two endpoints lying in adjacent hexagons can ev-dominate every point in those two hexagons.*

Proof. It follows from the fact that there will be an edge between any two points that lie within the same hexagon since the distance between them is at most 1. Therefore, an edge $e \in EVDS$ whose endpoint lies in that hexagon will ev-dominate every other point in that hexagon. \square

The outline of the algorithm is as follows. Initialize the set S^{ev} (which will hold the edges of EVDS) initially as empty. Now, arbitrarily pick an edge $e \in E$ whose endpoints lie in different cells. Add this edge to S^{ev} and set $E = E \setminus \{e\}$. Mark all points that are ev-dominated by e . If there are any unmarked vertices, now choose an edge $e \in E$ that is incident to any of the unmarked vertices, with its other endpoint lying in a different cell. Add e to S^{ev} and mark all the unmarked points that are ev-dominated by e . Repeat this process until every point in V is marked (see Algorithm 5).

Theorem 6.19. *Algorithm 5 gives a factor 5-approximation for EVDS problem on a UDG in $O(m + n)$ time.*

Proof. Algorithm 5 picks an edge e arbitrarily whose endpoints lie in different hexagons and then repeatedly selects an edge between an unmarked vertex and another vertex in the different hexagon until there are no unmarked vertices (see Figure 6.5). In Figure 6.5, we can observe that Algorithm 5 selected EVDS as $\{e_1, e_2, e_3, e_4, e_5\}$ whose cardinality is five whereas the optimal solution may have a single edge that will ev-dominate every given point (see the edge e in Figure 6.5). Next, one can see that the algorithm may select at most five times the optimal value, since an edge between points in two adjacent hexagons may ev-dominate the points in all of its adjacent eight hexagons. As we look at every edge between points to know whether its endpoints

Algorithm 5: Edge-vertex domination.**Input:** An UDG $G = (V, E)$ placed over an hexagonal grid.**Output:** An EVDS of G .Initialize $S^{ev} = \emptyset$, $E' = E$, and let all vertices in V be unmarked initially.**while** there is an edge $e \in E'$ with its both end-points unmarked **do**| Pick an edge $e \in E'$ such that $e = (u, v)$, the unmarked vertices $u \in A$ and $v \in B$, where
| A and B are adjacent hexagonal cells| Set $S^{ev} = S^{ev} \cup \{e\}$ and $E' = E' \setminus \{e\}$ | Mark all vertices which are incident to $N_e[e]$ **end****while** there is a hexagon containing unmarked vertices **do**| Pick any arbitrary edge $e \in E$ with at least one endpoint in that hexagon| $S^{ev} = S^{ev} \cup \{e\}$ **end****return** S^{ev}

are the marked vertices and select an edge at lines 3 and 8 of Algorithm 5, the running time is polynomial in m and n .

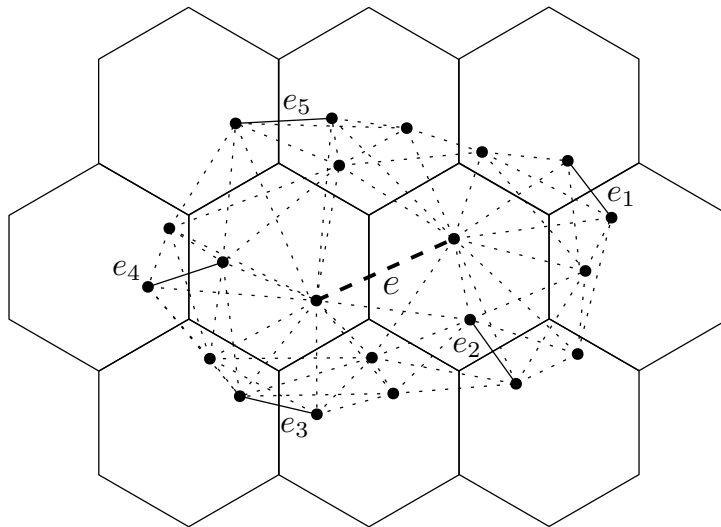


FIGURE 6.5: $EVDS = \{e_1, e_2, e_3, e_4, e_5\}$, and minimum $EVDS = \{e\}$.

The approximation factor five of Algorithm 5 follows due to the following two facts:

1. If both the endpoints of an edge e selected by Algorithm 5 lie within the same hexagon, then none of the vertices corresponding to these points are adjacent to a vertex of its adjacent hexagons.
2. Otherwise, an edge e selected by Algorithm 5 ev-dominates all the points in both the adjacent hexagons (Lemma 6.18).

All the cells (hexagons) in \mathcal{R} can be grouped as a collection of mega-cells (as in Figure 6.6), where each mega-cell consists of ten adjacent hexagonal cells (cells colored with the same color in

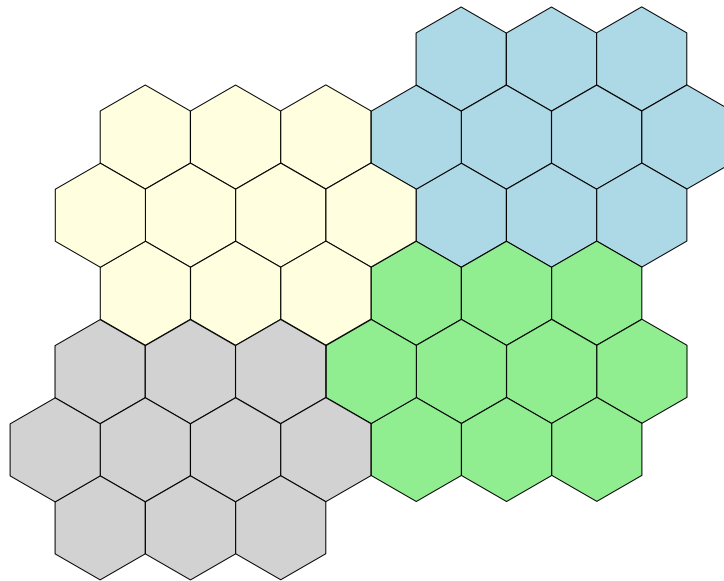


FIGURE 6.6: Four adjacent Mega-cells.

Figure 6.6). Algorithm 5 picks at most five edges to ev-dominate all the points in each mega-cell, whereas in the optimal solution, at least one edge is required. \square

6.4 Conclusion

In this chapter, we showed that the decision version of the EVDS on UDG is **NP-complete**. Later, we showed that the EVDS problem on UDG admits a PTAS, which utilizes the concept of an p -separated collection of subsets. Finally, we have provided a linear-time 5-factor approximation algorithm.

The results in this chapter are submitted for publication [72].

Chapter 7

Conclusion and Future Work

In this thesis, we studied different variants of the obnoxious facility dispersion problems in the Euclidean plane. These problems are called obnoxious facility location or geometric dispersion problems in the literature. They are studied by researchers of both the operations research domain and computer science domain (theoretical computer science). The researchers have usually modeled the facilities as disks in the Euclidean plane while solving these problems. Many authors have solved the different variants of the generalized versions of these problems and proposed exponential time algorithms [2, 5, 43] or approximation algorithms [9, 62, 63] since these generalized problems are notoriously hard and belong to the class of NP-hard problems. Hence, these algorithms are practical only for small values of k , where k is the number of facilities to be located. In this thesis, we studied several variants of these problems when restricted to line, circle, and convex polygon. We showed that most of these restricted problems can be solved exactly in polynomial time. Firstly, we studied the obnoxious facility location problem restricted to a line segment such as *constrained obnoxious facility location problem on a line segment* (COFL), *min-sum obnoxious facility location on a line-segment* (MOFL) and also when it is restricted to a circle named as *circular constrained obnoxious facility location* (CCOFL). Then, we investigated several variants of *semi-obnoxious facility location* (SOFL) problems such as *constrained semi-obnoxious facility location* (CSOFL) problem, SOFL on t -lines and *discrete semi-obnoxious facility location* (DSOFL) problem. We also studied another variant of the dispersion problem when there are no demand points in the plane named as *discrete k -dispersion on a convex polygon* (DKCONP) problem. Finally, we studied a variant of the dominating set problem in *unit disk graphs* known as *edge-vertex dominating set* (EVDS) problem, which can be modeled as an edge facility location problem.

We have presented a $(1 - \epsilon)$ -approximation algorithm for the COFL problem, with a runtime of $O((n \log n + k) \log \frac{\|pq\|}{2^{(k-1)\epsilon}})$. Here, $\epsilon > 0$ and $\|pq\|$ denotes the length of \overline{pq} . Additionally, we have introduced two exact polynomial-time algorithms: one utilizing a binary search on all candidates

and the other employing the parametric search technique of Megiddo [53]. These algorithms require $O((nk)^2)$ and $O((n \log n + k)^2)$ time, respectively. Notably, for the case of $k = 2$, an improved parametric technique enables us to devise an algorithm with a runtime of $O(n \log^2 n)$. Furthermore, we have demonstrated that the $(1 - \epsilon)$ -approximation algorithm presented for the COFL problem can be easily adapted to solve the circular variant of the problem with an additional multiplicative factor of n in the running time. Lastly, we solved the weighted min-sum variant of the problem in $O(n^3 k)$ time.

A few concluding remarks which would trigger further investigation in the context of the COFL problem:

- Parallel algorithm for DCOFL(P, k, L): Although we have provided a parallel algorithm for DCOFL($P, 2, L$), it remains unclear how to design a parallel algorithm for DCOFL(P, k, L) for any k . Exploring the development of such parallel algorithms could potentially improve the running time of the parametric search approach for the COFL problem.
- Unconstrained continuous OFL problem: For $k \geq 2$, the unconstrained (where the facility centers are not restricted to any particular domain such as line, circle, etc.) a continuous OFL problem has remained an elusive open problem since its initial mention by Katz et al. [43]. However, in the rectilinear variant of the problem, an $O((N/4)^k \log N \log \log N)$ -time algorithm is available [2], where $N = O(n + k)$. Investigating this open problem and seeking polynomial-time algorithms for $k = 1, 2$ would be valuable future endeavors.

For the problem of locating k semi-obnoxious facilities constrained to a line (CSOFL) when the given demand points have positive and negative weights. Specifically, we solved the problem of locating k semi-obnoxious facilities on a line to locate facilities with the maximum weight of the covered demand points in $O(n^4 k^2)$ time. Subsequently, we improved the running time to $O(n^3 k \cdot \max(n, k))$. Furthermore, we addressed two special cases of the problem where points do not have arbitrary weights. We showed that these two special cases can be solved in $O(n^3 k \cdot \max(\log n, k))$ time. For the first case, when $k = 1$, we also provide an algorithm that solves the problem in $O(n^3)$ time, and subsequently, we improve this result to $O(n^2 \log n)$. We also studied the SOFL for t -lines and showed that it can be solved in polynomial time but with a high order degree in t . Further, we investigated the complexity of discrete semi-obnoxious facility location (DSOFL) for the given candidate locations in convex position, and we showed that this problem can also be solved in polynomial time.

Following are some of the open problems that are worth considering in the context of the SOFL problem as future work:

- The continuous unrestricted variant of the semi-obnoxious facility location problem: given two sets (red and blue) of demand points with positive and negative weights (respectively)

in the plane and an integer k . The objective is to maximize the sum of the weights of the points covered by the union of k congruent non-overlapping disks of minimum radius centered anywhere in the plane (i.e., disks (facilities) may be centered anywhere in the plane).

- The discrete unrestricted variant of the semi-obnoxious facility location problem: given two sets (red and blue) of points with positive and negative weights (respectively), a set of candidate facility locations in the plane, and an integer k . The objective is to maximize the sum of the weights of the points covered by the union of k congruent non-overlapping disks of minimum radius centered at some of the candidate facility locations.
- To investigate the scenario when the disks are centered on the boundary of a convex polygon instead of a horizontal line or at vertices of convex polygon.
- To investigate the scenario when the disks are restricted to be centered at the grid points of a $t \times t$ grid in the plane.
- Finding a better than $O(n^2 \log n)$ time algorithm for the MAXBLUE-NORED problem for $k = 1$.

For the DKCONP problem, first, we gave an exact fixed-parameter algorithm with a runtime of $O(2^k n^2 \log^2 n)$. Second, is an exact polynomial time algorithm with a $O(n^4 k^2)$ runtime for any $k > 0$. We note that the general Euclidean k -dispersion problem remains open in terms of both polynomial-time approximation algorithms with better factors than $\frac{1}{2}$ and the design of exact fixed-parameter algorithms. No inapproximability bound for the Euclidean k -dispersion problem is known, unlike the metric k -dispersion problem which is not approximable with a factor better than $\frac{1}{2}$ unless $P = NP$. Further research is required to make progress in these directions.

Lastly, we also studied the complexity and approximability of the *edge-vertex dominating set* problem on unit disk graphs (EVDS-UDG). We first proved that the decision version of the EVDS-UDG is NP-complete. We then showed that the EVDS-UDG admits a PTAS. We also gave a simple 5-factor approximation algorithm in linear time. Although this 5-factor approximation algorithm is significantly faster when compared to PTAS, it requires a geometric representation of the input graph, whereas the proposed PTAS does not, hence is robust.

Chapter 8

Summary

Chapter 1: **Introduction.** introduces facility location problems and different variants of these problems.

Chapter 2: **Literature Review.** This chapter discusses the existing research on similar problems and compares our work with previous studies.

Chapter 3: Constrained Obnoxious Facility Location on a Line Segment. In this chapter, we first provided the formal definition of the decision version of the COFL problem, denoted as $\text{DCOFL}(P, k, L)$, and present a linear time algorithm based on a greedy approach for solving the decision version. Subsequently, we present an FPTAS (Fully Polynomial-Time Approximation Scheme) for the COFL problem, utilizing doubling search and bisection methods. This FPTAS involves multiple invocations of $\text{DCOFL}(P, k, L)$. Then, we discuss two exact algorithms for the COFL problem with polynomial time complexity. The first algorithm is based on the binary search and runs in $O((nk)^2)$ time. The second algorithm is based on the parametric search and runs in $O((n \log n + k)^2)$ time. Additionally, we introduced a faster algorithm based on the improved parametric search for $k = 2$, which achieves a time complexity of $O(n \log^2 n)$. Subsequently, we proceed to examine an FPTAS for the circular variant of the COFL problem, referred to as CCOFL. Finally, we present a dynamic programming-based polynomial time algorithm for solving the MOFL problem, which is a weighted min-sum version of the COFL. Then, we concluded the chapter.

Chapter 4: Semi-Obnoxious Facility Location on a Line. In this chapter, we first introduced and discussed various notations that will aid in comprehending the subsequent sections. Subsequently, we explored multiple configurations considered for computing the set of candidate radii, denoted as \mathcal{L}_{CAN} . Next, we delved into the transformation of the CSOFL problem into a minimum weight k -link path problem, given a candidate radius $\lambda \in \mathcal{L}_{\text{CAN}}$ that helps in solving the CSOFL problem exactly in $O(n^4 k^2)$ time. Additionally, we presented an improved

dynamic programming-based solution, which runs in $O(n^3k \cdot \max(n, k))$ time. Subsequently, we examined two special cases of the CSOFL problem, namely the ALLBLUE-MINRED problem and the MAXBLUE-NORED problem. These cases involve only two sets of weighted points. We show that these problems can be solved in $O(n^3k \cdot \max(\log n, k))$ time, and we also discussed the MAXBLUE-NORED problem for $k = 1$. Then, we extended the result of CSOFL to t -lines instead of a single line. Finally, we discussed a discrete variant of SOFL when candidate facility sites are in convex position, and we concluded the chapter by giving final remarks.

Chapter 5: Max-Min k -Dispersion for the Points in Convex Position. In this chapter, first, we discussed various notations that will help in understanding subsequent sections in this chapter. We discussed an exact fixed-parameterized algorithm for DKCONP problem by defining and solving its decision version, identifying the set of candidate radii, and invoking the decision algorithm for candidate radii by doing a binary search on the set of candidate radii. Later, we presented an exact polynomial-time algorithm with a time complexity of $O(n^4k^2)$. This algorithm utilizes two well-known concepts in computational geometry called Voronoi diagrams and Delaunay triangulation, which internally uses dynamic programming. Finally, we gave a linear time $\frac{1}{2\sqrt{2}}$ -approximation for $k = 3$ by exploiting the elementary geometry properties, and then we concluded the chapter.

Chapter 6: Edge-Vertex Domination in UDG. In this chapter, first, we showed that the decision version of the EVDS on UDG is NP-complete by describing a polynomial time reduction from the vertex cover problem, which is known to be NP-complete in planar graphs with maximum degree 3. Later, we showed that the EVDS problem on UDG admits a PTAS, which utilizes the concept of an p -separated collection of subsets. Finally, we gave a linear time 5-factor approximation algorithm, and then we concluded the chapter.

Chapter 7: Conclusion and Future Work. In this chapter, we presented concluding remarks summarizing the key findings and contributions discussed in the preceding sections. Additionally, we identify several open problems that could serve as potential avenues for future research.

Bibliography

- [1] VP Abidha and Pradeesha Ashok. “Geometric separability using orthogonal objects”. In: *Information Processing Letters* 176 (2022), p. 106245.
- [2] Shimon Abrevaya and Michael Segal. “Maximizing the number of obnoxious facilities to locate within a bounded region”. In: *Computers & operations research* 37.1 (2010), pp. 163–171.
- [3] Pankaj K Agarwal, Micha Sharir, and Sivan Toledo. “Applications of parametric searching in geometric optimization”. In: *Journal of Algorithms* 17.3 (1994), pp. 292–318.
- [4] Alok Aggarwal, Baruch Schieber, and Takashi Tokuyama. “Finding a minimum weight K-link path in graphs with Monge property and applications”. In: *Proceedings of the ninth annual symposium on Computational geometry*. (1993), pp. 189–197.
- [5] Toshihiro Akagi, Tetsuya Araki, Takashi Horiyama, Shin-ichi Nakano, Yoshio Okamoto, Yota Otachi, Toshiki Saitoh, Ryuhei Uehara, Takeaki Uno, and Kunihiro Wasa. “Exact algorithms for the max-min dispersion problem”. In: *Frontiers in Algorithmics: 12th International Workshop, FAW 2018, Guangzhou, China, May 8–10, 2018, Proceedings*. Springer. (2018), pp. 263–272.
- [6] Tetsuya Araki and Shin-ichi Nakano. “Max–min dispersion on a line”. In: *Journal of Combinatorial Optimization* 44.3 (2022), pp. 1824–1830.
- [7] David Avis, Godfried T Toussaint, and Binay K Bhattacharya. “On the multimodality of distances in convex polygons”. In: *Computers & Mathematics with Applications* 8.2 (1982), pp. 153–156.
- [8] Sang Won Bae, Arpita Baral, and Priya Ranjan Sinha Mahapatra. “Maximum-width empty square and rectangular annulus”. In: *Computational Geometry* 96 (2021), p. 101747.
- [9] Christoph Baur and Sándor P Fekete. “Approximation of geometric dispersion problems”. In: *Algorithmica* 30 (2001), pp. 451–470.
- [10] Sergey Bereg, Ovidiu Daescu, Marko Zivanic, and Timothy Rozario. “Smallest maximum-weight circle for weighted points in the plane”. In: *Computational Science and Its Applications—ICCSA 2015: 15th International Conference, Banff, AB, Canada, June 22–25, 2015, Proceedings, Part II*. Springer. (2015), pp. 244–253.

- [11] Therese Biedl and Goos Kant. “A better heuristic for orthogonal graph drawings”. In: *Computational Geometry* 9.3 (1998), pp. 159–180.
- [12] Razika Boutrig, Mustapha Chellali, Teresa W Haynes, and Stephen T Hedetniemi. “Vertex-edge domination in graphs”. In: *Aequationes mathematicae* 90.2 (2016), pp. 355–366.
- [13] Timothy M Chan and Konstantinos Tsakalidis. “Optimal deterministic algorithms for 2-d and 3-d shallow cuttings”. In: *Discrete & Computational Geometry* 56 (2016), pp. 866–881.
- [14] Richard L Church and Zvi Drezner. “Review of obnoxious facilities location problems”. In: *Computers & Operations Research* 138 (2022), p. 105468.
- [15] Richard L Church and Robert S Garfinkel. “Locating an obnoxious facility on a network”. In: *Transportation science* 12.2 (1978), pp. 107–118.
- [16] Richard Cole. “Slowing down sorting networks to obtain faster sorting algorithms”. In: *Journal of the ACM (JACM)* 34.1 (1987), pp. 200–208.
- [17] J Manuel Colmenar, Peter Greistorfer, Rafael Martí, and Abraham Duarte. “Advanced greedy randomized adaptive search procedure for the obnoxious p-median problem”. In: *European Journal of Operational Research* 252.2 (2016), pp. 432–442.
- [18] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, (2022).
- [19] João Coutinho-Rodrigues, Lino Tralhão, and Luís Alçada-Almeida. “A bi-objective modeling approach applied to an urban semi-desirable facility location problem”. In: *European journal of operational research* 223.1 (2012), pp. 203–213.
- [20] José Miguel Díaz-Báñez, Ferran Hurtado, Henk Meijer, David Rappaport, and Joan Antoni Sellarès. “The largest empty annulus problem”. In: *International Journal of Computational Geometry & Applications* 13.04 (2003), pp. 317–325.
- [21] Tammy Drezner, Zvi Drezner, and Anita Schöbel. “The Weber obnoxious facility location model: A big arc small arc approach”. In: *Computers & Operations Research* 98 (2018), pp. 240–250.
- [22] Tammy Drezner, Zvi Drezner, and Carlton H Scott. “Location of a facility minimizing nuisance to or from a planar network”. In: *Computers & Operations Research* 36.1 (2009), pp. 135–148.
- [23] Zvi Drezner and Zvi Drezner. *Facility location: a survey of applications and methods*. Springer, (1995).
- [24] Zvi Drezner and Horst W Hamacher. *Facility location: applications and theory*. Springer Science & Business Media, (2004).
- [25] Zvi Drezner and George O Wesolowsky. “Finding the circle or rectangle containing the minimum weight of points.” In: *Computers & Operations Research* (1994).

- [26] Zvi Drezner and George O Wesolowsky. “Obnoxious facility location in the interior of a planar network”. In: *Journal of Regional Science* 35.4 (1995), pp. 675–688.
- [27] H. Edelsbrunner and F.P. Preparata. “Minimum polygonal separation”. In: *Information and Computation* 77.3 (1988), pp. 218–232. ISSN: 0890-5401. DOI: [https://doi.org/10.1016/0890-5401\(88\)90049-1](https://doi.org/10.1016/0890-5401(88)90049-1).
- [28] Erhan Erkut. “The discrete p-dispersion problem”. In: *European Journal of Operational Research* 46.1 (1990), pp. 48–60.
- [29] Sandor Fekete. “On the complexity of min-link red-blue separation”. In: *Manuscript, department of applied mathematics, SUNY Stony Brook, NY* (1992).
- [30] Sándor P Fekete and Henk Meijer. “Maximum dispersion and geometric maximum weight cliques”. In: *Algorithmica* 38 (2004), pp. 501–511.
- [31] Greg N Frederickson and Donald B Johnson. “Generalized selection and ranking: sorted matrices”. In: *SIAM Journal on computing* 13.1 (1984), pp. 14–30.
- [32] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, (1979).
- [33] Mehraneh Gholami and Jafar Fathali. “The semi-obnoxious minisum circle location problem with Euclidean norm”. In: *International Journal of Nonlinear Analysis and Applications* 12.1 (2021), pp. 669–678.
- [34] Osman Gokalp. “An iterated greedy algorithm for the obnoxious p-median problem”. In: *Engineering Applications of Artificial Intelligence* 92 (2020), p. 103674.
- [35] Mehdi Golpayegani, Jafar Fathali, and Eiman Khosravian. “Median line location problem with positive and negative weights and Euclidean norm”. In: *Neural Computing and Applications* 24 (2014), pp. 613–619.
- [36] Mehdi Golpayegani, Jafar Fathali, and Haleh Moradi. “A particle swarm optimization method for semi-obnoxious line location problem with rectilinear norm”. In: *Computers & Industrial Engineering* 109 (2017), pp. 71–78.
- [37] Dorit S Hochbaum and Wolfgang Maass. “Approximation schemes for covering and packing problems in image processing and VLSI”. In: *Journal of the ACM (JACM)* 32.1 (1985), pp. 130–136.
- [38] John Hopcroft and Robert Tarjan. “Efficient planarity testing”. In: *Journal of the ACM (JACM)* 21.4 (1974), pp. 549–568.
- [39] Takashi Horiyama, Shin-ichi Nakano, Toshiki Saitoh, Koki Suetsugu, Akira Suzuki, Ryuhei Uehara, Takeaki Uno, and Kunihiro Wasa. “Max-min 3-dispersion problems”. In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 104.9 (2021), pp. 1101–1107.

- [40] Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. “Hamilton paths in grid graphs”. In: *SIAM Journal on Computing* 11.4 (1982), pp. 676–686.
- [41] Ramesh K Jallu, Sangram K Jena, and Gautam K Das. “Liar’s domination in unit disk graphs”. In: *Theoretical Computer Science* 845 (2020), pp. 38–49.
- [42] Sangram K Jena and Gautam K Das. “Vertex-edge domination in unit disk graphs”. In: *Discrete Applied Mathematics* (2021).
- [43] Matthew J Katz, Klara Kedem, and Michael Segal. “Improved algorithms for placing undesirable facilities”. In: *Computers & Operations Research* 29.13 (2002), pp. 1859–1872.
- [44] Yasuaki Kobayashi, SI Nakano, Kei Uchizawa, Takeaki Uno, Yutaro Yamaguchi, and Katsuhisa Yamanaka. “Max-min 3-dispersion on a convex polygon”. In: *37th European Workshop on Computational Geometry*. (2021).
- [45] B Krishnakumari, YB Venkatakrisnan, and Marcin Krzywkowski. “On trees with total domination number equal to edge-vertex domination number plus one”. In: *Proceedings-Mathematical Sciences* 126.2 (2016), pp. 153–157.
- [46] Balakrishna Krishnakumari, Mustapha Chellali, and Yanamandram B Venkatakrisnan. “Double vertex-edge domination”. In: *Discrete Mathematics, Algorithms and Applications* 9.04 (2017), p. 1750045.
- [47] Jason Robert Lewis. “Vertex-edge and edge-vertex domination in graphs”. PhD thesis. Clemson University, Clemson, (2007).
- [48] Ross D MacKinnon and GM Barber. “A New Approach to Network Generation and Map Representation: The Linear Case of the Location-Allocation Problem”. In: *Geographical Analysis* 4.2 (1972), pp. 156–168.
- [49] Costas D Maranas and Christodoulos A Floudas. “A global optimization method for Weber’s problem with attraction and repulsion”. In: *Large scale optimization: State of the art* (1994), pp. 259–285.
- [50] de Berg Mark, Cheong Otfried, van Kreveld Marc, and Overmars Mark. *Computational geometry algorithms and applications*. Springer, (2008).
- [51] Dániel Marx and Michał Pilipczuk. “Optimal parameterized algorithms for planar facility location problems using Voronoi diagrams”. In: *Algorithms-ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*. Springer. (2015), pp. 865–877.
- [52] Sourav Medya, Arlei Silva, Ambuj Singh, Prithwish Basu, and Ananthram Swami. “Group centrality maximization via network design”. In: *Proceedings of the 2018 SIAM International Conference on Data Mining*. (2018), pp. 126–134.
- [53] Nimrod Megiddo. “Applying parallel computation algorithms in the design of serial algorithms”. In: *Journal of the ACM (JACM)* 30.4 (1983), pp. 852–865.

- [54] Emanuel Melachrinoudis and Zaharias Xanthopoulos. “Semi-obnoxious single facility location in Euclidean space”. In: *Computers & Operations Research* 30.14 (2003), pp. 2191–2209.
- [55] Pawan K Mishra, SV Rao, and Gautam K Das. “Dispersion problem on a convex polygon”. In: *Information Processing Letters* 187 (2025), p. 106498.
- [56] Joseph SB Mitchell. *Approximation algorithms for geometric separation problems*. Tech. rep. State University of New York at Stony Brook, (1993). URL: <http://www.ams.sunysb.edu/~jsbm/papers/sep-2-10-94.pdf>.
- [57] Tim Nieberg and Johann Hurink. “A PTAS for the minimum dominating set problem in unit disk graphs”. In: *International Workshop on Approximation and Online Algorithms*. Springer. (2005), pp. 296–306.
- [58] Joseph O’Rourke. *Computational geometry in C*. Cambridge university press, (1998).
- [59] Joseph O’rourke, S Rao Kosaraju, and Nimrod Megiddo. “Computing circular separability”. In: *Discrete & Computational Geometry* 1 (1986), pp. 105–113.
- [60] Subhabrata Paul and Keshav Ranjan. “On vertex-edge and independent vertex-edge domination”. In: *International Conference on Combinatorial Optimization and Applications*. Springer. (2019), pp. 437–448.
- [61] Kenneth W Peters Jr. *Theoretical and algorithmic results on domination and connectivity (Nordhaus-Gaddum, Gallai type results, max-min relationships, linear time, series-parallel)*. Clemson University, (1986).
- [62] Zhongping Qin, Yinfeng Xu, and Binhai Zhu. “On some optimization problems in obnoxious facility location”. In: *Computing and Combinatorics: 6th Annual International Conference, COCOON 2000 Sydney, Australia, July 26–28, 2000 Proceedings 6*. Springer. (2000), pp. 320–329.
- [63] Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri Kumar Tayi. “Heuristic and special case algorithms for dispersion problems”. In: *Operations Research* 42.2 (1994), pp. 299–310.
- [64] Abdulgani Sahin and Bünyamin Sahin. “Total edge–vertex domination”. In: *RAIRO-Theoretical Informatics and Applications* 54 (2020), p. 1.
- [65] Bünyamin Şahin and Abdulgani Şahin. “Double Edge–Vertex Domination”. In: *International Conference on Intelligent and Fuzzy Systems*. Springer. (2020), pp. 1564–1572.
- [66] Michael Segal. “Placing an obnoxious facility in geometric networks”. In: *Nord. J. Comput.* 10.3 (2003), pp. 224–237.
- [67] Michael Ian Shamos. *Computational geometry*. Yale University, (1978).
- [68] Weiping Shang and Xiaodong Hu. “The roman domination problem in unit disk graphs”. In: *International conference on computational science*. Springer. (2007), pp. 305–312.

- [69] Vishwanath R Singireddy and Manjanna Basappa. “Constrained Obnoxious Facility Location on a Line Segment.” In: *33rd Canadian Conference on Computational Geometry*. (2021), pp. 362–367.
- [70] Vishwanath R. Singireddy and Manjanna Basappa. “Dispersing Facilities on Planar Segment and Circle Amidst Repulsion”. In: *Algorithmics of Wireless Networks*. Ed. by Thomas Erlebach and Michael Segal. Cham: Springer International Publishing, (2022), pp. 138–151.
- [71] Vishwanath R Singireddy and Manjanna Basappa. “Dispersing facilities on planar segment and circle amidst repulsion”. In: *Journal of Global Optimization* 88.1 (2024), pp. 233–252.
- [72] Vishwanath R Singireddy and Manjanna Basappa. “Edge-Vertex Dominating Set in Unit Disk Graphs”. In: *arXiv preprint arXiv:2111.13552 and submitted to Theory of Computing Systems, Springer* (2024).
- [73] Vishwanath R Singireddy, Manjanna Basappa, and NR Aravind. “Line-Constrained k -Semi-Obnoxious Facility Location”. In: *arXiv preprint arXiv:2307.03488 and submitted to International Journal of Computational Geometry & Applications (IJCGA)* (2024).
- [74] Vishwanath R. Singireddy, Manjanna Basappa, and Joseph S. B. Mitchell. “Algorithms for k -Dispersion for Points in Convex Position in the Plane”. In: *Submitted to Discrete Applied Mathematics (DAM), Elsevier*. (2023).
- [75] Vishwanath R Singireddy, Manjanna Basappa, and Joseph SB Mitchell. “Algorithms for k -Dispersion for Points in Convex Position in the Plane”. In: *Algorithms and Discrete Applied Mathematics: 9th International Conference, CALDAM 2023, Gandhinagar, India, February 9–11, 2023, Proceedings*. Springer. (2023), pp. 59–70.
- [76] Arie Tamir. “Obnoxious facility location on graphs”. In: *SIAM Journal on Discrete Mathematics* 4.4 (1991), pp. 550–567.
- [77] Alejandro Teran-Somohano and Alice E Smith. “Locating multiple capacitated semi-obnoxious facilities using evolutionary strategies”. In: *Computers & Industrial Engineering* 133 (2019), pp. 303–316.
- [78] Leslie G Valiant. “Universality considerations in VLSI circuits”. In: *IEEE Transactions on Computers* 100.2 (1981), pp. 135–140.
- [79] Yanamandram B Venkatakrisnan and Balakrishna Krishnakumari. “An improved upper bound of edge–vertex domination number of a tree”. In: *Information Processing Letters* 134 (2018), pp. 14–17.
- [80] Yanamandram B Venkatakrisnan and H Naresh Kumar. “On the algorithmic complexity of double vertex-edge domination in graphs”. In: *International Workshop on Algorithms and Computation*. Springer. (2019), pp. 188–198.
- [81] Andrea Wagner. “A new duality based approach for the problem of locating a semi-obnoxious facility”. In: (2015).

-
- [82] Da-Wei Wang and Yue-Sun Kuo. “A study on two geometric location problems”. In: *Information processing letters* 28.6 (1988), pp. 281–286.
- [83] Bowei Zhang. “Efficient Algorithms for Obnoxious Facility Location on a Line Segment or Circle”. In: *arXiv preprint arXiv:2210.07146* (2022).
- [84] Radosław Ziemann and Paweł Żyliński. “Vertex-edge domination in cubic graphs”. In: *Discrete Mathematics* 343.11 (2020), p. 112075.
- [85] Paweł Żyliński. “Vertex-edge domination in graphs”. In: *Aequationes mathematicae* 93.4 (2019), pp. 735–742.

List of Publications

Conferences

- Vishwanath R. Singireddy, Manjanna Basappa and Joseph S. B. Mitchell, "Algorithms for k -Dispersion for Points in Convex Position in the Plane", in proceedings of the 9th *Annual International Conference on Algorithms and Discrete Applied Mathematics (CALDAM 2023)*, Gandhinagar, India, Feb 9–11, *LNCS* 13947, pp. 59–70, 2023. https://doi.org/10.1007/978-3-031-25211-2_5
- Vishwanath R. Singireddy and Manjanna Basappa, "Dispersing Facilities on Planar Segment and Circle Amidst Repulsion", in proceedings of the 18th *International Symposium on Algorithmics of Wireless Networks (ALGOSENSORS 2022)*, Potsdam, Germany, Sep 8–9, *LNCS* 13707, pp. 138–151, 2022. https://doi.org/10.1007/978-3-031-22050-0_10
- Vishwanath R. Singireddy and Manjanna Basappa, "Constrained Obnoxious Facility Location on a Line Segment", in proceedings of the 33rd *Canadian Conference on Computational Geometry (CCCG 2021)*, Halifax, Nova Scotia, Canada, Aug 10–12, pp. 362–367, 2021. <https://projects.cs.dal.ca/cccg2021/wordpress/wp-content/uploads/2021/08/CCCG2021.pdf>

Journals

- Vishwanath R. Singireddy and Manjanna Basappa, "Dispersing Facilities on Planar Segment and Circle Amidst Repulsion", in *Journal of Global Optimization*, 88(1), pp. 233–252, 2024. <https://doi.org/10.1007/s10898-023-01303-x>
- Vishwanath R. Singireddy, Manjanna Basappa and Joseph S. B. Mitchell, "Algorithms for k -Dispersion for Points in Convex Position in the Plane", *Discrete Applied Mathematics (DAM)*, Elsevier, 2023 (Under Review)
- Vishwanath R. Singireddy and Manjanna Basappa, "Complexity and Approximability of Edge-Vertex Domination in UDG", *Theory of Computing Systems (TOCS)*, Springer, (2024) (Submitted)

- Vishwanath R. Singireddy, Manjanna Basappa and N. R. Aravind, "Line-Constrained k -Semi-Obnoxious Facility Location", *International Journal of Computational Geometry & Applications (IJCGA)*, 2024 (Submitted)

Biographical Sketch

Candidate's Biography

S Vishwanath Reddy is currently a research scholar in the Department of Computer Science and Information Systems at Birla Institute of Science and Technology, Pilani, Hyderabad Campus. He has completed his Bachelor of Technology (B.Tech) in Computer Science Engineering from JNTUH (MITS), Hyderabad and Master of Technology (M.Tech) in Computer Science Engineering from JNTUH (SITS), Hyderabad. His main areas of research are computational geometry and exact and approximation algorithms.

Supervisor's Biography

Dr. Manjanna B. received his Ph.D. from the Indian Institute of Technology, Guwahati, India, in 2016. He obtained his Master of Technology (M.Tech) degree from the National Institute of Technology Karnataka (NITK), Surathkal, India, and his Bachelor of Engineering (B.E.) degree from the University of Visvesvaraya College of Engineering, Bangalore, India. From August 2019 to July 2023, he served as an Assistant Professor in the Department of Computer Science and Information Systems at the Birla Institute of Technology and Science (BITS) Pilani, Hyderabad Campus, Hyderabad, India. He is currently an Assistant Professor in the Department of Computer Science and Engineering at the National Institute of Technology Karnataka (NITK), Surathkal, Mangalore. His research interests include algorithms and computational geometry.