INVESTIGATIONS ON TWO CLASSES OF COVERING PROBLEMS

MARK THOM
Bachelor of Science, University of Lethbridge, 2006
Bachelor of Science, University of Lethbridge, 2010
Master of Science, University of British Columbia, 2012

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

DOCTOR OF PHILOSOPHY

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

INVESTIGATIONS ON TWO CLASSES OF COVERING PROBLEMS

MARK THOM

Date of Defense: December 21, 2016

Dr. Robert Benkoczi Supervisor	Associate Professor	Ph.D.
Dr. Daya Gaur Committee Member	Professor	Ph.D.
Dr. Saurya Das Committee Member	Professor	Ph.D.
Dr. Ehab S. Elmallah Committee Member	Professor	Ph.D
Dr. Howard Cheng Department Chair	Associate Professor	Ph.D

Dedication

To Stephanie Renee McIntyre.

Abstract

Covering problems fall within the broader category of facility location, a branch of combinatorial optimization concerned with the optimal placement of service facilities in some geometric space. This thesis considers two classes of covering problems. The first, Covering with Variable Capacities (CVC), was introduced in [1] and adds a notion of capacity to the classical Uncapacitated Facility Location problem. That is, each facility has a fixed maximum quantity of clients it can serve. The objective of each variant of CVC is either to serve all clients, the greatest number of clients possible, or all clients using the least number of facilities possible. We provide approximation algorithms, and in a few select cases, optimal algorithms, for all three variants of CVC.

The second class of covering problems is barrier coverage. When the purpose of coverage is surveillance rather than service, a cost effective approach to the problem of intruder detection is to place sensors along the boundary, or barrier, of the surveilled region. A barrier coverage is complete when any intrusion is sure to be detected by some sensor. We limit our consideration of barrier coverage to the one-dimensional case, where the region is a line segment. Sensors are themselves line segments, whose span forms a detection range. The objective of barrier coverage as considered here is to form a complete barrier coverage while minimizing the total movement cost, the sum of the weighted distances moved by each sensor in the solution. We show that, by assuming the sensors lie in initial positions where their detection ranges are disjoint from the barrier, one-dimensional barrier coverage can be solved with an FPTAS. Along the way to developing the FPTAS, we give a fast, simple 2-approximation algorithm for weighted disjoint barrier coverage.

Acknowledgments

Deepest thanks to my friends and family for their companionship, humor, free catering services, and countless harangues for me to leave academia and get a real job.

I also wish to thank Dr. Robert Benkoczi, for his patient encouragement and mentorship, and Drs. Amir Akbary, Daya Gaur, Greg Martin, Nathan Ng, Shelly Wismath, Soroosh Yazdani, and Robert Benkoczi once more, for their generous financial and academic support throughout the past eight years.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

This dissertation addresses problems in combinatorial optimization, specifically those in the field of facility location. Facility location is concerned with the optimal placement of service facilities in some space, usually under various given constraints. For instance, a franchise owner may wish to choose a location for a new store in a large urban area according to various factors determining the desirability of the prospective neighborhood. These factors may include pedestrian traffic, the risk of robbery or theft in the area according to crime statistics, and so on.

Facility location problems are often demonstrated to be quantifiably hard, more frequently in the sense of being among the NP-hard problems. A well known question in theoretical computer science asks if any NP-hard problem admits an efficient algorithmic solution, an efficient algorithm being one that computes the best solution in polynomial time. From decades of experience, it is commonly believed that the answer is no. Based on the unlikelihood of finding an efficient solution to an NP-hard problem, one can instead design an approximation algorithm, which computes a feasible solution within a guaranteed range of the value of the best solution.

There are various types of facility location problems. Problems of median type concern the location of facilities in such a way that the total distance of all clients to the nearest facility is minimized. Problems of center type can be similarly phrased, by shifting the focus to the minimization of the maximum distance of clients to the facility. Conversely, problems of competitive type seek to claim clients from competitors' facilities by placing facilities that provide better service [16], [21].

This dissertation considers members of a specific family of facility location problems, covering problems on the line and in the two dimensional Euclidean plane. Some of these problems are in the NP-hard class while the complexity of others is unknown. There are many variations on covering problems, but they are generally defined according to the following template.

Objects in fixed locations on a line segment or in the plane (clients) are in need of coverage by service providers (facilities), which may be located in fixed or variable positions on the line segment or plane. The objective of covering problems is to maximize the value of the problem metric, a measure which numerically quantifies the quality of service provided by a placement of facilities under the constraints imposed by the problem. The higher the value of the metric, the better the service.

The choice of metric varies according to the covering problem, and helps to determine the character of the problem. If the metric states that we must cover all clients within range of a prescribed set of facilities at minimum total cost of opening each facility, then we are dealing with a set cover problem. Another example is found in the metric that counts the number of clients served as the result of opening n facilities. It expresses exactly the goal of maximizing the number of clients served with a given number of facilities. This type of problem is broadly classified as a maximum cover facility location problem. If each facility can serve only a fixed number of clients, then we face a capacitated cover facility location problem, in either the set cover or maximum setting.

Our thesis investigates the circumstances in which two classes of covering problems admit constant-factor approximation algorithms. At times we isolate to edge cases exhibiting structure not found in the general case. This is done to circumvent conditions in which constant-factor approximation algorithms have been shown unlikely to exist.

1.2 Motivations

The choice of the geometric space in which facilities and clients are represented often decides whether there is likely to exist an optimal algorithm that solves the problem in time polynomial in the size of the problem. Such problems are said to be optimally solvable in polynomial time. In industry, problems in wireless services, surveillance and intruder detection frequently arise as covering problems. Since facility location problems in industry often concern placements in physical space, industrial interest in covering problems is usually bracketed to those occurring in two or three dimensional spaces.

The covering problem was introduced in [11] and has been widely used in practice in areas such as the location of emergency vehicles, of retail facilities, and of telecommunications equipment. However, the simple "all or nothing" covering constraint has been found to be too restrictive for many applications, and several relaxations have been proposed and studied in the last decade. The survey [7] presents three relaxations: (a) the gradual cover model where the degree with which a client is served decreases as its distance to the facility increases; (b) the cooperative cover model where several facilities can contribute to serving the same client; (c) the variable covering radius model where the planner can choose the covering range for the facilities, but the opening cost for the facility increases with its range. The authors of [1] introduce a new family of covering problems, Covering with Variable Capacities (CVC), which addresses the client coverage problem in the presence of interference in wireless networks. Arguably, solutions to problems in the CVC models have immediate applications in the mobile telephony industry.

When the purpose for coverage is surveillance rather than service, a cost effective approach is to monitor the perimeter of the area in order to detect intruders. In one-dimensional coverage problems, the barrier is represented by a horizontal line segment and sensors are initially placed on the line containing the line segment. The goal is to compute new positions for some subset of the given sensors, ensuring that every point in the barrier segment is within the detection range of some sensor. In typical two dimensional coverage problems, sensors are represented as points in the plane and assigned a radius of coverage while the barrier is represented as the boundary of some closed planar region.

For a large scale deployment, using only the minimum number of sensors needed to achieve full coverage is key to reducing costs. This discourages random deployments of sensors, as they are unlikely to make efficient use of detection ranges, resulting in few redundant sensors in the typical random covering.

1.3 Objectives and Contributions

This dissertation summarizes and extends the work of [1] and [6] as follows.

- The NP-hardness of the uniform CVC problem is demonstrated (section 3.1). It originally appeared in [1].

- Approximation algorithms are provided for the uniform, maximum and set cover CVC problems (section 3.2). They all appeared in [1].

- Supposing a constant number of range/capacity pairs, an optimal polynomial time algorithm for the 1-dimensional maximum uniform CVC problem is given (chapter 4). The result is unpublished at the time of this writing, and was established by me.

- The weighted disjoint MinSum barrier coverage problem is introduced, and shown to be NP-hard (chapter 5 and section 5.1). It first appeared in [6] in its unweighted version. The idea for the FPTAS was developed in an afternoon of collaboration between myself and the other three authors of [6]. This lead to the proof of NP-hardness of unweighted disjoint MinSum by me, and the proof of the correctness of the FPTAS, a joint effort between myself and Zachary Friggstad.

- A 2-approximation algorithm for the weighted disjoint MinSum problem is developed (section 5.2). The algorithm was suggested to me by Robert Benkoczi and Daya Gaur, and I gave its correctness proof.

- The FPTAS of [6] is extended to the weighted disjoint MinSum problem using the 2-approximation algorithm developed here. In doing so, a quadratic factor improvement to the asymptotic time complexity of the original FPTAS is obtained in addition to a linear factor space complexity improvement. (sections 5.3 and 5.4). At the time of this writing, the weighted disjoint MinSum FPTAS is unpublished.

1.4 Methodologies

1.4.1 Linear Programming

A *linear program* is a way of representing an optimization problem in which the values of n real variables are to be determined under linear constraints, and under a linear optimization criterion.

The standard form of a linear program is as follows.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad 1 \leq i \leq m \\ & x_i \geq 0, \quad 1 \leq i \leq n \end{aligned}$$

The *objective function* decides the optimal value, and must be a linear function. The m constraints in the second row are expressed as linear inequalities that the variables must satisfy, where all a_{ij} and b_i are real constants.

If we define the matrix $A = [a_{ij}]_{1 \leq i \leq n, 1 \leq j \leq m}$ and the column vector $b = [b_i]_{1 \leq i \leq n}$, the linear program can be rewritten as

$$\begin{aligned} \text{minimize} \quad & cx \\ \text{subject to} \quad & Ax \geq b \\ & x_i \geq 0, \quad 1 \leq i \leq n \end{aligned}$$

where the inequality $Ax \geq b$ is interpreted as the componentwise inequality on the real components of the vectors Ax and b. We refer to this representation as the *matrix form* of a linear program.

A vector of non-negative components x satisfying $Ax \geq b$ is called a *feasible solution*, since it meets the constraints of the problem.

1.4.2 The Simplex Method

Due to the relative compactness and simplicity of the presentation, we limit our overview to the so-called *revised simplex method*.

We start with a linear program in matrix form. The idea of the simplex method is to gradually refine a feasible solution x until an optimal solution is obtained. At all times, x will have no more than m non-zero components.

The m non-zero components are known collectively as the *basis variables* in each iteration of the simplex method. By the definition of A, each column $a_{\_,i}$ of A corresponds to a unique vector component x_i. We denote as A_B the submatrix of A whose columns correspond to basis variables. Since there are m basis variables and m constraints, A_B is an $m \times m$ matrix. x_B will denote the vector of basis variables, and similarly with respect to A_N and x_N for the non-basis variables.

By adding a few extra variables, we can always tighten the constraints to $Ax = b$. If the linear inequalities are cast as planar inequalities in $\mathcal{R}^n$, then the space $Ax \geq b$ can be realized geometrically as a polytope in $\mathcal{R}^n$, with any feasible solution x as a point in the polytope. If the constraints are tightened to $Ax = b$, x becomes a point on the surface of the polytope.

The refinements of the simplex method consist of selecting a "surface point" x, which is pushed along an edge, as far as possible without causing it to leave the domain of the polytope. The direction it is pushed along will always decrease the value of the objective function. Once x cannot progress further, it is said to occupy an extreme point, a vertex of the polytope. At this time, one of the basis variables will be reduced to 0, and one of the non-basis variables will be positive. The positive non-basis variable will enter the basis, and the 0-valued basis variable will be removed from the basis. The process will continue in iterations until no further reduction to the objective function is possible, which is to say that the current feasible solution is an optimal one.

We begin by introducing m non-negative slack variables to the problem, $x_{n+1}, \ldots, x_{n+m}$, which we add to x as components. A becomes $[A| - I_m]$ where I_m is the $m \times m$ identity matrix, and the m constraints $Ax \geq b$ become $Ax = b$. Finally, we append a vector of m 0's onto the vectors c and b, to correspond to the new slack variable components of x.

As with A, we will refer to the basis subvectors of x and c as x_B and c_B, and similarly to the non-basis subvectors as x_N and c_N.

To illustrate the process of selecting an entering and leaving variable, we use matrix algebra. We have $Ax = A_B x_B + A_N x_N = b$. By definition of the basis, at the beginning of any iteration, $x_N = 0$, so that $z = cx = c_B x_B + c_N x_N = c_B x_B$.

We write x_B as $x_B = A_B^{-1} b - A_B^{-1} A_N x_N$ (chapter 7 of [12] contains an argument for the nonsingularity of A_B). Then $z = cx = c_B x_B + c_N x_N = c_B(A_B^{-1} b - A_B^{-1} A_N x_N) + c_N x_N = c_B A_B^{-1} b + (c_N - c_B A_B^{-1} A_N) x_N$.

Let $w = c_N - c_B A_B^{-1} A_N$, and suppose it has a negative component. If we increase the corresponding component of x_N by a positive amount, it is clear from the above equation for z that z can only decrease, even while potentially causing the basis variables to change in order to preserve $Ax = b$. The component is increased until one or more of the basis variables is valued at 0. The first basis variable observed to reach 0 leaves the basis, swapping places with the component variable, which is now positively valued.

If w has no negative component, z is an optimal solution and the revised simplex method terminates.

1.4.3 Duality

Any linear program has a dual program, obtained as follows. For each of the m constraints of the form $\sum_{j=1}^{n} a_{ij}x_j \leq b_i$, we introduce a new variable, y_i, and add the constraint $y_i \geq 0$. We get the constraints

$$yi\Big(\sum_{j=1}^{n} a_{ij}x_j\Big) \leq b_i y_i$$

for each $1 \leq i \leq m$. Summing the inequalities gives

$$\sum_{i=1}^{m} yi\Big(\sum_{j=1}^{n} a_{ij}x_j\Big) \leq \sum_{i=1}^{m} b_i y_i$$

We note that the inequality holds for all feasible solutions of the linear program $\{x_i\}_{i=1}^{n}$ and any vector of non-negative components $\{y_i\}_{j=1}^{m}$ we may choose.

We add constraints to the dual linear program using the objective function coefficients c_j. We obtain n constraints of the form

$$\sum_{i=1}^{m} a_{ij}y_i \leq c_j \forall 1 \leq j \leq m$$

The dual linear program is summarized as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{m} b_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{m} a_{ij}y_i \leq c_j, \quad 1 \leq j \leq n \\ & y_i \geq 0, \quad 1 \leq i \leq m \end{aligned}$$

Rearranging inequalities as before, we find that

$$\sum_{i=1}^{m} b_i y_i \leq \sum_{i=1}^{m}\Big(\sum_{j=1}^{n} a_{ij}x_j\Big)y_i \leq \sum_{j=1}^{n} x_j\Big(\sum_{i=1}^{m} a_{ij}y_i\Big) \leq \sum_{j=1}^{n} c_j x_j$$

We have proved the *theorem of weak duality*. The *theorem of strong duality* says that the optimal solution values of any feasible linear program and its dual coincide.

Chapter 2

Literature Review

2.1 Covering with Variable Capacities

CVC generalizes the classical capacitated covering due to [23] where an upper bound on the total demand that can be served by a facility is imposed. Facilities correspond to wireless base stations employing omni-directional antennas and clients represent service subscribers. It is assumed that the location of the clients is given. Demands (bandwidth requirements) and profits (revenue) are associated with the clients.

In the CVC model of [6], every facility has a variable covering range and the facilities need to be located and assigned a covering range. The range can only be increased at the expense of the capacity, as increasing the power of the radio transmitter causes more interference in the network [8], [18], [19].

The maximum and set cover CVC formulations of [1] were at the time of publication open problems, aimed at understanding the connection between interference and base station capacity in networks utilizing the popular code division multiple access (CDMA) technology. Several researchers have investigated the idea of using this dependency to improve the performance of networks. For example, the authors of [24] show that there are significant savings in resource utilization for wide band CDMA networks when the range of the base stations is appropriately chosen. The authors of [25] describe a cellular network that exploits this phenomenon.

We should note that CVC abstracts away from some of the finer details of certain models. For wireless transmissions, the data rate received by a mobile user is affected by her proximity to the serving base station, as well as the amount of received interference power.

The authors of [1] describe three models of CVC problems: CVC with fixed facilities (or simply CVC) where the location of the facilities is given and the objective is to maximize the total profit of the clients served, maximum CVC where a set of clients with maximum total profit must be covered by a fixed number of facilities, and set cover CVC where the entire set of clients must be covered by a set of facilities with total minimum cost.

2.2 Barrier coverage problems

Kumar *et al.* [20] were first to formalize the barrier coverage by k sensors (k-barrier coverage). They established the equivalence between k-barrier coverage and the problem of determining k-vertex disjoint paths between a pair of vertices in a graph. The paper has spurred research on many aspects of barrier coverage problems, varying from density estimates of random deployments [3] to the relaxation of coverage requirements suited for the study of localized algorithms [9].

The optimization problems defined by Czyzowicz *et al.* [14] [13] are one dimensional problems where the barrier is modeled by a line segment and sensors are initially located on the line containing the segment. The goal is to compute new positions for a subset of the sensors so that every point in the target line segment is within the sensing range of at least one sensor. Several objec- tive functions have been studied: minimizing the maximum distance (MinMax) traveled by one sensor [5], minimizing the number (MinNum) of sensors moved [10], and minimizing the total (MinSum) travel distance [4].

In [14] and [13], Czyzowicz *et al.* introduced objective functions centered around the minimization of cost spiking quantities and studied the one dimensional barrier coverage problem. The MinMax problem minimizes the maximum distance travelled by any sensor; the MinNum problem minimizes the number of sensors used.

In [10], Chen *et al.* show the one-dimensional MinMax problem to be solvable in $O(n^2 \log n)$ time, where n is the number of sensors. It is possible to derive more efficient algorithms under the assumption that the sensing ranges are uniform. In [13], Czyzowicz *et al.* separate the barrier problem with uniform sensing ranges into two cases, those where the total length of sensing ranges exceeds the size of the line segment making up the barrier, and those where it does not. They provide exact algorithms for either case, with time complexity $O(n)$ for the second case and time complexity $O(n^2)$ for the first. Andrews and Wang [2] improved the $O(n^2)$ bound to $O(n \log n)$ for the case of uniform sensors. An extension of the line coverage problem where each point on the barrier is to be covered by k-sensors (k-line coverage) was studied recently by Wang *et al.* [26], who gave optimal algorithm for the case of uniform sensors.

Versions of two-dimensional MinMax and MinSum problems are addressed in [15], in which there are multiple barriers fixed as line segments in the plane. Sensors can have arbitrary sensing ranges and be located anywhere in the plane, and both Euclidean and rectilinear metrics of distance are considered. Variants of both problem types are introduced and distinguished according to the number of barriers to be covered. As well, the orientation of barriers plays a role, as barriers may be oriented parallel with or perpendicular to one another. While in general sensors may move freely, some variants require sensors to move to the closest point on a barrier. Dobrev *et al.* [15] demonstrate exact algorithms for MinMax and MinSum in the single barrier and k parallel barrier cases, and show that in all remaining cases, their variants of MinMax and MinSum are NP-hard.

Surprisingly, the combinatorial structure of the MinSum problem is not yet completely understood. Czyzowicz *et al.* [14] proved the NP-hardness for the general problem with non-uniform sensing ranges. We note that their proof constructs a MinSum instance where the initial position for some of the sensors is inside the target line segment. The proof also indicates that constant factor approximations for the general MinSum problem are not possible unless P=NP. Except for this inapproximability result, the only restricted instances solved are those with uniform sensors, for which an exact algorithm with time complexity $O(n)$ for the case of $R < L$ and an $O(n^2)$ exact algorithm for the case $R \geq L$ are possible.

Of chief interest to us is the MinSum problem, which minimizes the total distance travelled by all sensors to their final positions in the solution. A common variation on the MinSum problem is the addition of some notion of sensor heterogeneity. In practice, some sensing units may consume less power or have greater sensing ranges, while newer units simply haven't had the time to suffer the effects of wear, deterioration and obsolescence. Therefore, barrier coverage models often allow for associated differences in the effectiveness of individual sensors to be expressed.

One model exhibiting sensor heterogeneity is found in Bar-Noy *et al.* [4]. They devise a sensing model where each sensor has finite battery life, and both moving and sensing cause sensors to consume battery power. Covering therefore occurs in two phases: the deployment phase, where sensors are moved into their final positions, and the covering phase, where sensors decide their sensing ranges to form a full barrier coverage. The amount of power consumed by movement and sensing is determined by given equations, which together establish the lifetime of each sensor. The goal is to maximize the barrier coverage lifetime, which is defined as the minimum lifetime of any sensor used in the coverage. For the case of sensors with variable sensing ranges, they [5] give an FPTAS to minimized the total energy, and an FPTAS to minimize the maximum energy spent. For the case of fixed sensing ranges they give an inapproximability result.

In [6], we restrict the MinSum problem to the case where initially, sensors may only lie in positions where their detection ranges are disjoint from the barrier. Their model allows for sensing ranges to be non-uniform. They show the resulting *DisjointMinSum* problem to be NP-hard, and present a fully polynomial time approximation scheme (FPTAS) for *DisjointMinSum*, proving that solutions with approximation ratio arbitrarily close to 1 can always be computed for it in polynomial time. This result provides a new direction of investigation given that Czyzowicz *et al.* [14] proved that the unrestricted MinSum problem admits no constant factor approximation.

A generalization of DisjointMinSum unexplored in [6] is obtained by assigning individual weights to the movement costs of sensors, a second dimension of sensor heterogeneity after allowing ranges to be non-uniform. The objective of WeightedDisjointMinSum is to minimize the total weighted distance of sensors travelled in the solution under the constraints of DisjointMinSum.

Chapter 3

Covering with Variable Capacities

Given a set of client locations in a Euclidean space, the covering facility location problem is concerned with determining a set of optimal locations for facilities required to service the clients. Facilities come with a built-in range, or radius, of coverage. In order for a client to be serviced by a facility, it must fall within the radius of the facility.

Covering problems have been used in many industrial applications, among them the placement of emergency services, retail locations and cellular towers. It was found that the simple "all or nothing" model in which a client is either serviced totally or not at all is too restrictive for many practical purposes. Therefore, a number of relaxations of all or nothing have been proposed in the literature. They can be camped into three broad categories:

- *gradual cover*: the quality of service received by each client is numerically quantified, and gradually declines according to its distance from the nearest facility;
- *cooperative cover*: multiple facilities assist in providing services to individual clients simultaneously;
- *variable covering radius*: the planner chooses the radius of coverage for each facility; the greater the range of coverage, the greater the facility cost.

In this chapter we discuss a family of covering problems distinct from all of these, called Covering with Variable Coverings (CVC). CVC is a generalization of the classical capacitated covering problem, where each facility has a predetermined cap on the number of clients it can serve. As in the capacitated covering problem, we suppose that the client set is fixed and given to us as an input.

Each client has a demand, represented as a positive integer, to be met by a single facility, and offers a profit, also represented as a non-negative integer, in exchange for coverage. Facilities have a pre-determined set of range/capacity pairs. The planner selects a distinct range/capacity for each facility, under the restriction that the total demand of clients covered cannot exceed the selected capacity of the facility, and must fall within the selected range. As the range of coverage increases, capacity declines.

These constraints are motivated by considerations in the field of mobile telephony. In the configuration of a cellular tower, interference increases with service range, and consequently, coverage capacity goes down. Similarly, some clients may have greater bandwidth requirements than others, and the service provider may wish to charge them more.

CVC problems can be situated in any Euclidean space. The variants of CVC studied here as follows.

Problem 1 (**CVC**).

Input:

- A set $\mathcal{C} = \{c_i : i \in I\}$ of clients where I is the index set of clients.
- For each client c_i, a non-negative integral demand d_i and profit p_i.
- A set $\mathcal{F} = \{f_j : j \in \mathcal{J}\}$ of facilities where $\mathcal{J}$ is the index set of facilities.
- For each facility f_j, a set R_j of allowed ranges and for each $r \in R_j$, a corresponding capacity c_{jr}. By N_{jr} we denote the set of clients within the covering range r of facility f_j.

Output:

- For each facility f_j, a range $r_j \in R_j$.

- For each facility f_j, a subset of clients $I_j \subseteq I$ serviced only by f_j, satisfying $\sum_{i \in I_j} d_i \leq c_{jr_j}$ and $c_i \in N_{jr_j}$ for all $i \in I_j$.

Objective:

- To maximize the total profit of clients served, $\max_{i \in \bigcup_{j \in \mathcal{J}} I_j} p_i$.

Problem 2 (**Maximum CVC**).

Input:

- Same as for Problem 1. The set of facilities now represent *candidate* locations, of which a subset must be chosen.
- A positive integer k.

Output:

- k facilities to be opened, indexed by $\mathcal{J}^* \subseteq \mathcal{J}$, where $|\mathcal{J}^*| = k$.
- For each facility f_j, for $j \in \mathcal{J}^*$, a range $r_j \in R$.
- For each facility f_j, $j \in \mathcal{J}^*$, a subset of clients $I_j \subseteq I$ serviced only by f_j, satisfying $\sum_{i \in I_j} d_i \leq c_{jr}$ and $c_i \in N_{jr}$ for all $i \in I_j$.

Objective:

- To maximize the total profit of clients served, $\max_{i \in \bigcup_{j \in \mathcal{J}} I_j} p_i$, using k or fewer facilities.

Problem 3 (**Set Cover CVC**).

Input:

- Same as for Problem 1. The set of facilities now represent *candidate* locations, of which a subset must be chosen.

Output:

- A set of facilities to be opened, indexed by $\mathcal{J}^* \subseteq \mathcal{J}$.
- For each facility f_j, for $j \in \mathcal{J}^*$, a range $r_j \in R$.
- For each facility f_j, $j \in \mathcal{J}^*$, a subset of clients $I_j \subseteq I$ serviced only by f_j, satisfying $\sum_{i \in I_j} d_i \leq c_{jr}$ and $c_i \in N_{jr}$ for all $i \in I_j$. Additionally, all clients are served, so that $\bigcup_{j \in \mathcal{J}^*} I_j = I$.

Objective:

- To minimize the number of opened facilities, $\min |\mathcal{J}^*|$.

The uniform version of a CVC problem is a special case describing any of the above three problems where for every client c_i, we have $d_i = p_i = 1$. We note that CVC generalizes the capacitated covering problem, which can be considered otherwise identical to CVC, but with the restriction that every facility uses the same range/capacity pairing.

We begin by demonstrating that the uniform CVC with fixed facilities problem is NP-hard in the plane, even when the number of range/capacity pairs is limited to 2. We will show that the formulation of CVC as a compact integer program yields a large integrality gap.

Then, we give $1/2 - \varepsilon$ approximation algorithms for both CVC and maximum CVC problems with no restrictions on client demands or profits.

Lastly, we give a simple linear programming formulation of set cover CVC, and introduce a simple rounding scheme that finds good approximate solutions in practice. We conjecture but do not prove that the approximation has an integrality gap of $e/(e-1)$ where e is Euler's number.

3.1 NP-completeness of the uniform CVC with fixed facilities problem

We show that uniform CVC is NP-complete, even when all fixed facilities are limited to the same two covering range/capacity pairs. To do this, we will reduce from the Var-Linked Planar 3-SAT problem [17].

A planar 3-SAT problem is a version of the classic 3-SAT problem in which a bipartite graph is formulated from a Boolean formula in conjunctive normal form (CNF). On one side, there is a vertex for each variable of the formula while on the other, there is a vertex for each clause. A variable vertex and a clause vertex are connected by an undirected edge if and only if either the variable or its negation are present in the clause. Finally, the bipartite graph is planar, meaning that it is possible to draw the graph in the plane in such a way that no two edges cross.

A var-linked planar 3-SAT problem (VLP 3-SAT) implements the definition of planar 3-SAT, but adds two conditions. First, suppose the CNF formula has n variables. They are put into a linear ordering $x_1, \ldots, x_n$. An edge (x_i, x_{i+1}) is added for every $i = 1, \ldots, n$, where $x_{n+1} = x_1$. The addition of these edges can be done so that the planarity of the graph is not violated, but clearly, the graph is no longer bipartite. Second, each variable x_i occurs in exactly three clauses, one in which it is negated, and two in which it is non-negated.

We show in the next theorem that any uniform CVC with fixed facilities problem in the plane, where facilities select from up to same two capacity/range pairs, is reducible from var-linked planar 3-SAT.

Theorem 3.1. *The uniform CVC problem with fixed facilities in the plane is NP-complete, even when the facilities use the same two ranges with capacities in the set* $\{1,2\}$.

Proof. We consider the CVC problem in the form of a decision problem. That is, given some value P, is there a feasible assignment of clients to facilities such that the total profit of the serviced clients is at least P? Once a solution is determined, it is easy to check whether its total profit meets or exceeds P, so we can say that the problem is in NP.

The idea of the reduction is summarized as follows. First, we draw a planar representation of the clause-variable edges of the graph, with the variables linked by edges in the ordered cycle, as described in the definition of the problem. The drawing is modified, so that each variable vertex is substituted with a construct of three vertices, which we call a *variable gadget*. The fact that the original planar graph has a cycle connecting all its variable vertices ensures that we can place variable gadgets without crossing any edges. Similarly, every edge of the original planar graph is replaced with a *path gadget*.

A geometric CVC problem is built by selecting facilities and clients from among the gadget nodes, and assigning ranges and capacities to the facilities. We will show that the original Boolean CNF formula is satisfiable if and only if the CVC problem constructed has a covering in which the attainable total profit is at least P, where P is the number of client vertices contained in the transformed graph.

To obtain a variable gadget, we replace every variable vertex y_i with a path of three vertices, x_i, ξ_i, and $\overline{x_i}$, and consider that x_i and $\overline{x_i}$ are facilities while ξ_i is a client. For every clause C_j, we add an edge connecting x_i to the clause vertex representing C_j if C_j contains x_i in its nonnegated form. Similarly, we connect $\overline{x_i}$ and C_j with an edge if C_j contains x_i in its negated form. The vertex ξ_i is connected only to x_i and $\overline{x_i}$, as the middle vertex of the path that forms the variable gadget.

The modified planar graph obtained in this way resembles the one illustrated in Figure 3.1. The white vertices correspond to the literal facilities x_i and $\overline{x_i}$. The black vertices are the clients ξ_i (which are connected exclusively to white vertices) and the clause clients C_j.

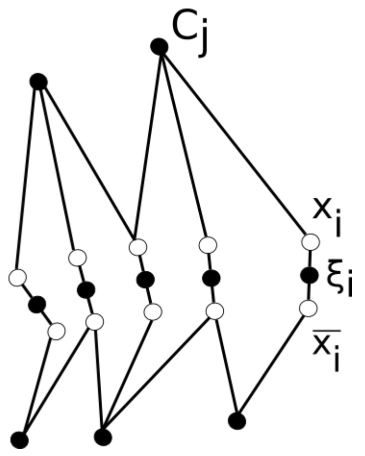

Figure 3.1: Part of a transformed planar graph.

We now claim that these additions can be made to the graph without introducing any edge crossings. To prove it, let $\delta_D > 0$ be the shortest Euclidean distance between a node and an edge not incident to it in the drawing of the original graph. We place x_i and $\overline{x_i}$ apart from the middle vertex ξ_i at distance $\delta < \delta_D$.

We observe that in the original drawing, there are three edges emanating from C_j to clause j's variable vertices that do not cross the edges of the cycle linking the variable vertices. Therefore, there is always a suitable orientation of the path connecting x_i, ξ_i and $\overline{x_i}$ waiting to be found. As a client, ξ_i will only be coverable by facility x_i or facility $\overline{x_i}$, due to range constraints we will introduce later. We will refer to x_i and $\overline{x_i}$ as *literal facilities* from now on.

Next, additional clients are introduced, one for each formula clause C_j. These we refer to as the *clause clients*. We connect each clause client with the literal facilities corresponding to the three literals of the clause, using a path gadget. A path gadget is an alternating sequence of clients and facilities beginning with a client and ending with a facility. Neighboring nodes in the path gadget are separated by a distance of δ. Each facility in the path gadget has available to it only one range/capacity pair, of range δ and capacity 1. It is therefore capable of covering either its predecessor or successor client on the path, but not both at once. For the sake of brevity, we will refer to a facility at an end of the path as the path gadget *f-end* and the client at the other end as its *c-end*.

For each clause j, we will use three path gadgets to connect C_j to the literal facilities/-variables that make it up. The c-end of the first path gadget will be connected via a single edge to some literal facility x_i if the literal y_i appears in j. Recall that each literal occurs in exactly three clauses, one in its negated form and the other two in its non-negated form. The non-negated literal facilities/variables form the f-ends of two path gadgets each, connecting them to their containing clause clients C_j. The negated literal facilities/variables form the f-ends of one path gadget each, likewise connecting them to their containing clause clients C_j. In either case, we ensure that the length of edges connecting clause clients and literal facilities is $\varepsilon < \delta$. It is easy to see that the size of the resulting graph is polynomial in the size of the original VLP-3SAT graph. Let n be the number of vertices in the original

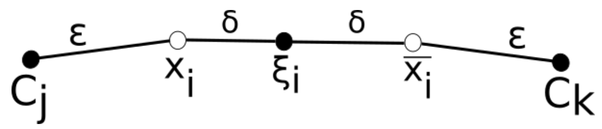

Figure 3.2: The lengths of edges connecting a variable gadget to the graph

.

clause-variable graph, which is to say that n is the number of variables and clauses in the CNF formula. The clause-variable graph can be drawn on an $(n-1) \times (n-1)$ grid in $O(n)$ time.

The shortest distance δ separating a node on the grid and a line segment on the grid can be taken to satisfy the bound $\Omega(1/n^2)$. Each path gadget introduced has up to $O(n^3)$ client and facility vertices, and therefore the size of the resulting graph is $O(n^4)$.

In order to prove that what we have described is a reduction, we must show the Boolean CNF formula to be satisfiable if and only if the CVC problem obtained from it has maximum profit at least P. In the reduction proof, P is the total number of clients represented in the transformed graph.

Suppose the boolean assignment $y_1, \ldots, y_n$ satisfies each clause of the formula. Set the radii of the variable gadget facilities x_i and $\overline{x_i}$ as follows.

$$r_{x_i} = \begin{cases} \delta & \text{if } y_i = 0 \\ \varepsilon & \text{if } y_i = 1 \end{cases} \qquad\qquad r_{\overline{x_i}} = \begin{cases} \delta & \text{if } y_i = 1 \\ \varepsilon & \text{if } y_i = 0 \end{cases}$$

This way, client ξ_i is coverable by either facility x_i or $\overline{x_i}$. In particular, x_i covers ξ_i if and only if y_i is false.

So, fix a literal y_i from the satisfying assignment and consider whether y_i is true. If it is, the literal facility x_i has its covering range set to ε with capacity 2, following the conditional definition of r_{x_i}. It is used to cover its two containing clause vertices, to which it is connected by edges of length ε. Its counterpart facility at $\overline{x_i}$ has range δ, and it is used to cover ξ_i.

If instead y_i is false, then the literal facility $\overline{x_i}$ has covering range ε, and it is used to cover its sole containing clause vertex. This time, x_i is assigned the greater range δ, and it is used to cover ξ_i.

From the vantage point of the clients, each ξ_i is covered by some facility in the above scheme, since every pair of facilities x_i and $\overline{x_i}$ contains a facility corresponding to a false literal. For a clause client C_j, one of its neighbouring literal facilities evaluates to true, and hence is assigned to cover C_j, whether the literal is in negated or non-negated form. Therefore, the existence of a satisfying assignment allows us to construct a complete cover under the given capacity/range constraints.

Conversely, we must show that a complete cover corresponds to a satisfying assignment. From the constraints we put on the lengths of the graph edges, we note that each ξ_i is covered by exactly one of its two neighbouring literal facilities. If x_i covers ξ_i, we assign x_i false, and if $\overline{x_i}$ covers ξ_i, we assign it true. Since exactly one of x_i and $\overline{x_i}$ covers ξ_i, the assignment is consistent.

Similarly, each clause client C_j is covered by some x_i. From the structure of the transformed graph, it follows that the covering literal facility of each clause C_j corresponds to a literal contained in the clause j. Since the literal determined by the covering literal facility is assigned true, the clause j is true in the assignment obtained. Therefore, we have determined a satisfying assignment from the complete covering. □

3.2 Algorithms for CVC with fixed facilities

We give three integer programming formulations for CVC with fixed facilities in an arbitrary metric space. We show that they all give large integrality gaps, even when clients are assumed to have uniform, unit demand. We note the similarity of CVC with fixed facilities to the separable assignment problem, and how it implies the existence of a $1-1/e$ approximation algorithm based on randomized rounding of a linear problem solution. We go on to discuss a $1/2$ approximation algorithm based on a simple greedy approach.

3.2.1 Compact integer programming formulations for CVC with fixed facilities

The integer problems discussed here are devised for CVC with fixed facilities, but can be easily adapted to handle set cover and maximum CVC.

Without loss of generality, we suppose that facility capacity is a decreasing function of covering range. Let k be the total number of clients and f the total number of facilities. The first IP is a natural formulation of the problem. The two types of variables are defined as follows.

$$x_{ua} = \begin{cases} 1 & \text{if client } u \text{ is served by facility } a \\ 0 & \text{otherwise} \end{cases} \qquad y_{ar} = \begin{cases} 1 & \text{if facility } a \text{ uses range } r \\ 0 & \text{otherwise} \end{cases}$$

An optimal solution of the first CVC covering problem on fixed facilities is given in the following IP. p_u is the profit of client u and d_u is its demand.

$$\begin{aligned}
\text{maximize} \quad & \sum_u p_u z_u \\
\text{subject to} \quad & \sum_{r \in R_a} y_{ar} \leq 1, \quad \forall 1 \leq a \leq f \\
& x_{ua} \leq z_u, \quad \forall 1 \leq u \leq k, 1 \leq a \leq f \\
& \sum_{a=1}^{f} x_{ua} \geq z_u, \quad \forall 1 \leq u \leq k \\
& x_{ua} \leq \sum_{r : u \in N_{ar}} y_{ar}, \quad \forall 1 \leq a \leq f, 1 \leq u \leq k \\
& \sum_{u=1}^{k} d_u x_{ua} \leq \sum_{r \in R_a} c_{ar} y_{ar}, \quad \forall 1 \leq a \leq f
\end{aligned}$$

The first constraint ensures that only one range is selected for each open facility. The second and third constraints ensure that every covered client u is covered by at least d_u facilities, so that its demand is satisfied; note that if the client is uncovered, z_u can be the set to 0, and the third constraint then has no effect. The fourth constraint ensures that if u is covered by facility a, it is covered at the range r configured for a. Finally, the fifth constraint ensures the number of clients covered by any facility a is not greater than the capacity provided at a's range.

Now we construct an example showing an integrality gap of $f-1$ in the IP. In the example, all client demands and profits are equal to 1. Let all f facilities be fixed at the same point F, and let the clients be positioned at one of two points, P and Q. P is an outermost point of the first covering range of each facility, of capacity m. Similarly, Q is the outermost point of the second, greater covering range of each facility, of capacity 1. We place m clients at P and $m(f-2)$ clients at Q.

It is easy to see that $m+f-1$ is the greatest number of clients that can be covered. In the optimal solution, a single facility covers the m clients at P, and the remaining $f-1$ facilities cover $f-1$ clients at Q. Alternatively, there is a fractional solution where $x_{ua} = \frac{1}{f}$ for all clients u and facilities a. If we take $y_{ar_1} = (f-1)/f$ and $y_{ar_2} = 1/f$ for all facilities a, we readily see that the first three constraints are satisfied. We are left to verify the fourth constraint. This is easy, as $\frac{m(f-1)}{f} \leq \frac{m(f-1)}{f} + \frac{1}{f}$. Since the value of the fractional solution is $m(f-1)$, and since this equals the total demand of all clients under the linear relaxation of the original IP, it is optimal. We therefore have an integrality gap of $\frac{m(f-1)}{m+f-1}$, which approaches $f-1$ as m becomes large.

The second integer program uses all the same variables and has more constraints, but is structurally simpler. Variable x_{ua} is interpreted in the same way, but the meaning of variable y_{ar} is broadened to indicate the selection of r or some range greater than r at facility a. If y_{ar} has its second index sorted in non-decreasing order of range, we have an IP with the constraints

$$y_{a,r} \geq y_{a,r+1}, \qquad \forall 1 \leq a \leq f, r < |R_a|$$

$$x_{ua} \leq y_{ar}, \qquad \forall 1 \leq u \leq k, 1 \leq a \leq f, r \text{ is the smallest range covering } u$$

$$\sum_{u=1}^{k} x_{ua} \leq c_a - \sum_{r>1} d_{ar} \cdot y_{ar}, \qquad \forall 1 \leq a \leq f$$

with the same objective function. The new coefficients d_{ar} represent the difference in capacity between ranges $r-1$ and r, so that $d_{ar} = c_{a,r-1} - c_{a,r}$. c_a represents the maximum capacity at facility a, at the first and thus smallest available range. In cooperation with the first constraint, the fifth constraint enforces the capacity limit of the facility at the selected range.

The third IP drops the y_{ar} variables, and adds the constraints

$$\sum_{s=1}^{k} x_{sa} \leq c_a - D_{ua} \cdot x_{ua}, \qquad \forall 1 \leq a \leq f, 1 \leq u \leq k$$

Here, D_{ua} is the loss of maximum capacity incurred if facility a serves client u. The furthest client served by a determines the maximum range, and hence the capacity of the client, reflected in the value of D_{ua}.

These formulations have a large integrality gap as well. The quadratic number of constraints places practical limits on the size of CVC problems that can reasonably hoped to be solved using IP solvers.

3.2.2 Known approximation results

Problem 1 is closely related to SAP, the Separable Assignment Problem. SAP concerns the packing of sized items into bins of varying capacities, where the subset of items packable into bin i is denoted I_i. Each item is assigned a value, and the goal to maximize the total value of packed items.

SAP was studied in [8], where it was shown to have two approximation algorithms, a linear program based $1-1/e$ approximation algorithm, and a $1/2$ approximation algorithm based on local search. From an instance of a CVC problem, we construct an instance of SAP as follows. Each facility is interpreted as a bin. Define $I_{i,r}$ the set of clients (items) that can be covered by facility (placed in bin i) at range r. Then $|I_{i,r}| \leq c_{ir}$. Finally, let $I_i = \bigcup_{r \in R} I_{i,r}$.

This reduction implies there is a $1-1/e$ LP rounding based approximation algorithm for the CVC problem. While the number of constraints is exponential, there is a separation oracle, allowing the relaxed linear program to be solved in polynomial time.

3.2.3 A greedy approximation algorithm for CVC with fixed facilities

Here we present a greedy algorithm for CVC with fixed facilities. It is well-known that there is a $(1-\varepsilon)$ FPTAS for the general Knapsack problem (see section 3.1 of [27]). Suppose that we can call the Knapsack FPTAS using the notation $K(\alpha, C, I)$, where C is the capacity of the Knapsack, I is the set of items to be covered along with their profits, and $0 < \alpha < 1$ is the desired approximation ratio. This means that the total profit of $K(\alpha, C, I)$ is at least $(1-\varepsilon) \cdot OPT$, where $\alpha = 1 - \varepsilon$ and OPT is the maximum attainable profit.

Algorithm 1 Solve the provided CVC problem using a greedy approximation technique.

```
1: procedure $V(\alpha)$
2:     for $j \in \mathcal{J}$ do
3:         for $r \in R_j$ do
4:             $(p_{jr}, I_{jr}) \leftarrow K(\alpha, c_{jr}, N_{jr})$        ▷ $p_{jr}$ is the profit, and $I_{jr}$ the chosen clients.
5:         $p_j \leftarrow \max_{r \in R_j} p_{jr}$        ▷ $p_j$ is the solution with largest profit.
6:         $I_j \leftarrow \arg\max_{I_{jr}} p_{jr}$
7:         $r_j \leftarrow \arg\max_{r \in R_j} p_{jr}$
8:         $I \leftarrow I \setminus I_j$
       return $\sum_{j \in \mathcal{J}} p_j$
```

The algorithm considers each facility by iterating through the facility index set $\mathcal{J}$. At each range/capacity pair that might be assigned to f_j, it views the assignment of clients to j as a Knapsack problem, where each item c_i has demand d_i and value p_i. It solves the Knapsack problem at integrality gap α, so that the total profit of clients selected is within a factor of α in proportion to the maximum profit attainable there. Then, it selects the coverage offering the greatest profit out of all the range/capacity choices, and removes the clients selected from I. This continues over every remaining facility, until the sum of profits obtained from every set of facility assignments is returned.

We demonstrate that Algorithm $V(\alpha)$ has an approximation bound of $\alpha/(\alpha+1)$.

Theorem 3.2. *Algorithm $V(\alpha)$ is an $\alpha/(\alpha+1)$-approximation algorithm for the CVC problem with fixed facilities. It has runtime $O(fm^4/\varepsilon)$, where $\alpha = 1-\varepsilon$, f is the number of facilities, and m is the number of clients.*

Proof. Fix an optimal solution of the CVC problem and let Q_j be the set of clients assigned to facility f_j in the optimal solution but not available to the iteration considering f_j in the approximation algorithm, because they were selected in previous iterations. Let $A_j = OPT_j \setminus Q_j$, where OPT_j is the set of clients assigned to f_j in the optimal solution. If S is any set of clients S, we denote the total profits of clients in S is as p_S. Since K is an α-approximation algorithm for Knapsack, we have

$$\alpha p_{A_j} \leq p_{K(\alpha, c_{jr_j}, N_{jr_j})}$$

for the selected range $r \in R_j$. This follows from $A_j \subseteq OPT_j$ and the fact that every client of A_j was available to Algorithm $V(\alpha)$ at the time it assigned clients to f_j for $r \in R_j$.

Summing, we get

$$\alpha \sum_{j=1}^{|\mathcal{J}|} p_{A_j} \leq \sum_{j=1}^{|\mathcal{J}|} p_{K(\alpha, c_{jr_j}, N_{jr_j})}$$

Since every set of clients Q_j has its total profit claimed in both the optimal solution and the solution V computed by Algorithm $V(\alpha)$, we also have

$$\sum_{j=1}^{|\mathcal{J}|} p_{Q_j} \leq V$$

We have shown

$$OPT \leq \sum_{j=1}^{|\mathcal{J}|} p_{Q_j} + \sum_{j=1}^{|\mathcal{J}|} p_{A_j} \leq V\left(1 + \frac{1}{\alpha}\right)$$

For the run time analysis, it is well known that the standard Knapsack FPTAS has a run time of $O(m^3/\varepsilon)$, where m is the number of items and $\alpha = 1-\varepsilon$ for $\varepsilon > 0$. Since there are at most m different Knapsack instances considered with respect to each facility, and no more than f facilities, we arrive at the claimed bound. $\square$

For the uniform CVC problem with fixed facilities, we set $\alpha = 1$ and obtain a $1/2$-approximation algorithm by dispensing with the use of the Knapsack FPTAS in Algorithm $V(\alpha)$. Since demand is uniform, the optimal Knapsack solution is trivially obtained by selecting clients greedily, according to largest profit. Apart from the replacement of the Knapsack FPTAS with the much simpler, optimal greedy solution, the algorithm is identical to Algorithm $V(\alpha)$, as it its analysis.

Corollary 3.3. *The greedy approximation algorithm for uniform CVC with fixed facilities is a $1/2$-approximation algorithm when restricted to the uniform CVC problem with fixed facilities.*

To see that the bound is tight, consider the uniform CVC problem with fixed facilities on four points on the real number line, equally spaced, positioned at the integers 1 through 4. The even numbered points are facilities with a single range/capacity pair, $(1,1)$, while the odd numbered points are clients. Since the problem is uniform, clients have a profit and demand of 1. The greedy approximation will assign the facility at 2 to the client at 3. An optimal solution is to assign facility 2 to client 1 and facility 4 to client 3. Therefore, the problem exhibits a performance ratio of $1/2$.

3.2.4 Adapting the greedy algorithm to maximum CVC

Algorithm $V(\alpha)$ can be made to handle maximum CVC instances. The algorithm runs in k iterations, where k is the number of facilities to be opened. For each iteration, a knapsack instance for each remaining candidate facility is considered over each available range/capacity pair. The facility with the most profitable knapsack solution is opened at the witnessing range/capacity pair. The maximum CVC version of Algorithm $V(\alpha)$ is given in pseudocode as Algorithm 2.

We immediately see that the conditions of Theorem 3.2 are satisfied, and we obtain

Theorem 3.4. *There is a greedy algorithm that computes an $\alpha/(\alpha+1)$ approximate solution to the maximum CVC problem.*

Algorithm 2 Solve the provided maximum CVC problem using the greedy approximation technique.

```
1: procedure $V(\alpha)$
2:     $\mathcal{J}' \leftarrow \mathcal{J}$
3:     for $i$ from 1 to $k$ do
4:         for $j \in \mathcal{J}'$ do
5:             for $r \in R_j$ do
6:                 $(p_{jr}, I_{jr}) \leftarrow K(\alpha, c_{jr}, N_{jr})$   ▷ $p_{jr}$ is the profit, and $I_{jr}$ the chosen clients.
7:             $p_j^{(i)} \leftarrow \max_{r \in R_j} p_{jr}$   ▷ $p_j$ is the solution with largest profit.
8:             $I_j^{(i)} \leftarrow \arg\max_{I_{jr}} p_{jr}$
9:             $r_j^{(i)} \leftarrow \arg\max_{r \in R_j} p_{jr}$
10:        $j \leftarrow \arg\max_{j \in \mathcal{J}'} p_j^{(i)}$
11:        $p_j \leftarrow p_j^{(i)}$
12:        $I_j \leftarrow I_j^{(i)}$
13:        $I \leftarrow I \setminus I_j$
14:        $\mathcal{J}' \leftarrow \mathcal{J} \setminus \{j\}$
       return $\sum_{j \in \mathcal{J}} p_j$
```

3.3 A column generation approach for Set Cover CVC

Turning to Set Cover CVC, we give an approximation algorithm that proceeds in two rounds. From a natural integer program formulation of set cover CVC, we relax to a linear program, which we solve. The solution is fed to a simple randomized rounding procedure that yields good results in practice. We present strong evidence for the conjecture that the rounding procedure is a $e/(e-1)$-approximation scheme, where e denotes Euler's constant.

In the language of the classic weighted set cover problem, any covering set $\mathcal{S}$ is a facility instantiated over two features. The first is one of the available given range/capacity pairs. The second is a subset of clients within the selected range, whose total demand does not exceed the selected capacity. All facility instantiations S are collected in the set $\mathcal{S}$. The one time opening cost of a facility configuration S at a given range/capacity pair is its set weight, w_S. It is clear that the size of the set $\mathcal{S}$ is exponential in the size of the client set.

This scheme of enumerating all client assignments against every applicable configuration of facilities means that the integer program is exactly that of the classical weighted set cover. It is simply

$$\begin{aligned}
\text{minimize} \quad & \sum_{S \in \mathcal{S}} w_S x_S \\
\text{subject to} \quad & \sum_{S \in \mathcal{S} : S \ni c} x_S \geq 1, && \forall c \in \mathcal{C} \\
& x_S \in \{0,1\}, && j = 1, \ldots, |\mathcal{S}|
\end{aligned}$$

We obtain a linear program by applying the relaxation of the value constraints on x_S to $x_S \geq 0$ for all $S \in \mathcal{S}$. Due to the exponential size of $\mathcal{S}$ in the number of clients and facilities, we cannot hope to compute a solution of the LP to any reasonable standard of efficiency. Therefore, we apply the technique of column generation.

The column generation method is largely patterned after the revised simplex method, but with the added complication that not all columns of the constraint matrix are considered in each step. That is, in a linear program expressed in the form

$$\text{minimize} \sum_{j=1}^{nr} x_j$$

$$\text{subject to } Ax \leq b$$

$$\text{and } x \geq 0$$

it is assumed that A is fully known. In any given step of the revised simplex method, all columns j of A of capacity $c_j > 0$ are considered for shifting into the basis matrix B. In any step of column generation, B forms our total knowledge of A up to that point. After each iteration of the algorithm, a new column of A is generated and appended to B according to the notion of best reduced cost.

If we look to the linear program above, we see that we are dealing with a version of the set cover problem. A column of A describes a set of clients coverable by a given facility/radius pair, with the clients enumerated in the rows. Although the number of facility/radius pairs is independent of the number of clients, the addition of new clients to the plane increases the number of columns of A at an exponential rate. This greatly inflates the combinatorial complexity of the compact program in both its integral form and linear relaxation, which we overcome using column generation.

The column generation method initializes the constraint matrix B to the $m \times m$ identity matrix I_m, m the number of client points [12]. As in the revised simplex method, every column of B is a column of A, with the added caveats that B may contain multiple copies of the same column of A. The sets of clients and facilities are always the same, and the first radius in the nondecreasing sequence of radii belonging to each facility is always 0. Therefore, B begins life as a submatrix of A.

The column giving the smallest negative as it applies to B in any iteration is computed as follows. First, the solution to the dual problem

$$\begin{gathered}\text{maximize } \sum_{i=1}^{m} y_i \\ \text{subject to } [B^T y]_j \leq 1 \ \forall j \\ \text{and } y_i \geq 0 \ \forall i\end{gathered}$$

is obtained. If $a_{\text{-}j}$ is a column of A, then, following the revised simplex method, we want to select j satisfying $\text{argmin}_{j \in cols(A)} c_j - a_{\text{-}j} y < 0$.

We proceed by iterating over the columns of $M = [m_{ij}]$, whose columns identify the clients i within range of the facility/radius pair fixed to the column j. That is, $m_{ij} = 1$ if client i is within range of the facility/range pair j, and $m_{ij} = 0$ otherwise. For the j^{th} column of M, $m_{\text{-}j}$, we compute the Hadamard product $h = m_{\text{-}j} \circ y = [m_{ij} \cdot y_i]$ and find the largest c_j components of h. Initially, $a_{\text{-}j}$ is the zero vector. If k is the index of any of the largest c_j components in h, we set $a_{kj} = m_{kj}$. Since there are no more than c_j ones in $a_{\text{-}j}$ at the end of this process, from $a_{\text{-}j} \leq m_{\text{-}j}$ we see that the single facility assignment represented by $a_{\text{-}j}$ meets the range and capacity constraints of the problem.

The $a_{\text{-}j}$ minimizing $\operatorname{argmin}_{j \in cols(A)} c_j - a_{\text{-}j} y < 0$ is returned and appended to B, if such an $a_{\text{-}j}$ exists, and the algorithm continues to the next iteration. If such an $a_{\text{-}j}$ does not exist, then B is fully determined and the algorithm terminates with the solution of the linear program

$$\text{minimize} \sum_{j=1}^{cols(B)} x_j$$

$$\text{subject to } [Bx]_j \geq 1 \; \forall j$$

$$\text{and } 0 \leq x_j \leq 1 \; \forall j$$

We summarize the column generation procedure in the pseudocode of Algorithm 3. A

Algorithm 3 The column generation procedure.

```
1: $B \leftarrow I_m$
2: $num\_fr \leftarrow$ the number of facility/range pairs
3: while true do
4:     $y \leftarrow$ the solution of the dual program with constraint matrix $B$
5:     $perms \leftarrow$ random shuffle of the integers $\{1, 2, \ldots, num\_fr\}$
6:     for $k$ from 1 to $num\_fr$ do
7:         $i \leftarrow perms[k]$
8:         $(sorted, idx) \leftarrow sort(M_{\text{-},i} \circ y)$   ▷ Return the sorted vector, and the permutation
   of the indices of $M_{\text{-},i} \circ y$ made by $sort$.
9:         $s \leftarrow sum(sorted[1 : cap(i)])$   ▷ Sum the first $cap(i)$ components of $sorted$,
   where $cap(i)$ is the capacity of the facility/range pair at $i$.
10:        if $1 - s < 0$ then
11:            for $j$ from 1 to $cap(i)$ do
12:                $a_{i,idx[j]} \leftarrow M_{i,idx[j]}$
13:            break
14:    if $1 - y \cdot a \geq 0$ then
15:        break
16:    $B \leftarrow$ append column $a$ to $B$
```

number of alternatives to the above scheme were attempted, and found to underperform when implemented in GNU Octave. The first was a direct translation of the column generation algorithm described in [12].

From the above analysis, the optimal solution to the original linear program can be found using only the m' sets $S_1, \ldots, S_{m'}$ corresponding to the columns selected in the formation of B.

Now that we can efficiently obtain the solution to the original, exponential size linear program, we are left to contend with the fact that the solution is fractional. To get a feasible set cover out of the fractional solution, we use a simple randomized rounding scheme, described in pseudocode as Since every value of every variable x^*_S in the optimal solution

Algorithm 4 The randomized rounding procedure.

```
1: Let $\mathcal{S}$ be the set of generated columns.
2: Let $x^*_S \in \mathcal{S}$ stand for $S$ in the solution of the optimal linear program.
3: repeat
4:     for all $S \in \mathcal{S}$ do
5:         $x_S \leftarrow 1$ with probability $x^*_S$, otherwise $x_S \leftarrow 0$
6:         $F \leftarrow F \cup \{S : x_S = 1\}$
7: until $F$ is a set cover
8: for all $S \in \mathcal{S}$ in random order do
9:     if $F \setminus \{S\}$ is a set cover then
10:        $F \leftarrow F \setminus \{S\}$
   return $F$
```

is bounded between 0 and 1, we can interpret the solution as a probability distribution determining the likelihood of every set S being included in the cover. We perform a series of coin flips, one for each set S, using a biased coin with probability x^*_S of heads. If this does not produce a set cover, we start a new series of flips, and repeat until a set cover is produced.

Once we have a cover, its sets are considered in random order, where sets that are deemed superfluous are removed. A set is superfluous if its removal from F does not change the fact that F is a set cover.

The results of Algorithm 4 are shown in Figure 3.3. Nearly 500 instances of set cover CVC were generated at random in the plane. The number of clients is indicated in the x-axis. Each facility was assigned 5 covering ranges with values in the interval $(0,1)$ and five capacities from the set $\{1,\ldots,5\}$ determined uniformly at random, such that the capacity is a nonincreasing function of the range.

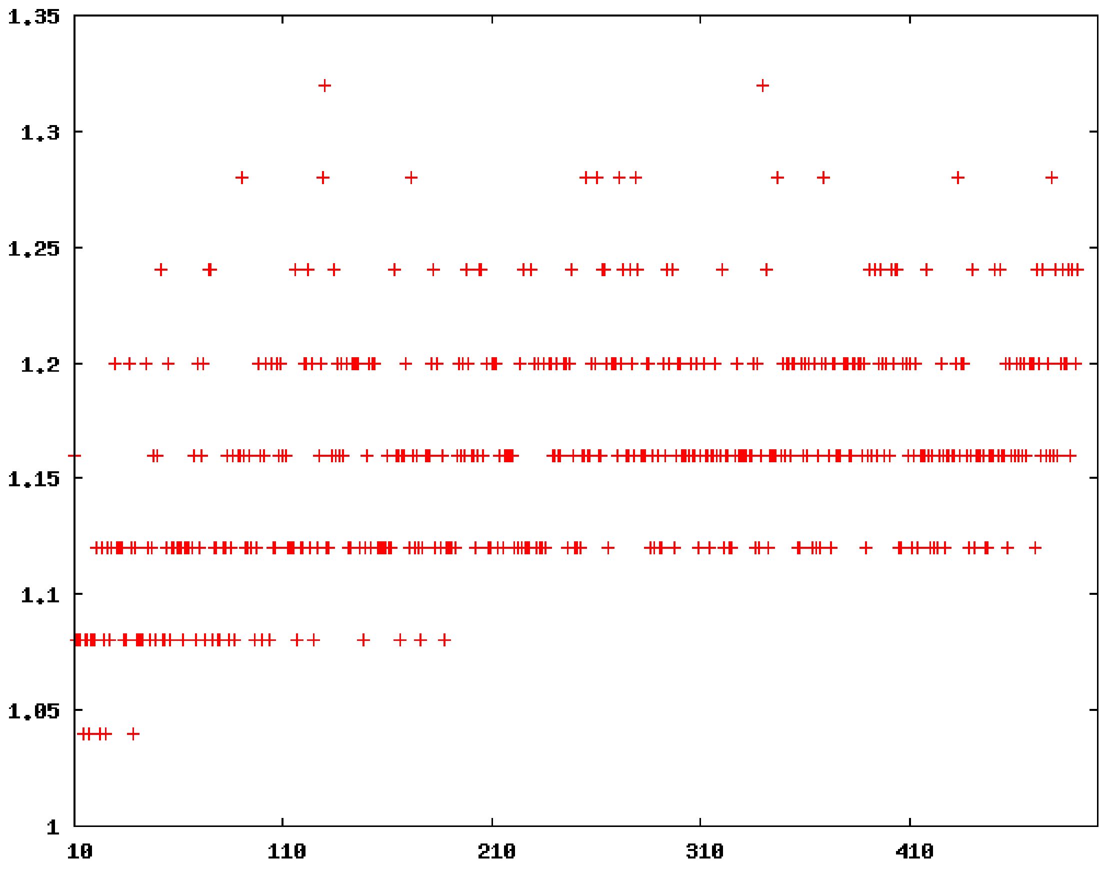

Figure 3.3: The results of the randomized rounding/compact IP experiment.

The *y*-axis indicates the performance ratio of Algorithm 4 to the optimal solution. Both Algorithm 4 and the optimal IP were implemented in GNU Octave.

Chapter 4

1-dimensional Maximum CVC with Uniform Demand

In this chapter, we give an optimal, polynomial time algorithm for maximum CVC on the line for the case where every client has uniform demand and profit, assuming a constant number of ranges. That is, every client u_i has profit $p_i = 1$ and demand $d_i = 1$, and is represented as a point on the line. Facilities can be placed anywhere on the line, and their range/capacity configurations are chosen from a common set of range/capacity pairs.

We begin by developing the results of section 4.1. After pointing out several parallels to the results of Mirchandani *et al.* in [22], we will see that our Algorithm 5 is a slightly restricted version of their Algorithm $DP2$. To make the parallels explicit, we will follow the terminology of [22]. Before giving the shared definitions, we describe the problem solved in the third section of [22], and its relation to maximum CVC on the line.

In the linear capacitated covering problem considered in [22], Mirchandani *et al.* place n facilities with fixed locations and capacities on the line. To each client, the problem assigns a non-empty interval of facilities capable of covering the client. A facility is a member of a client interval if and only if it can service that client.

In section 3 of [22], Mirchandani *et al.* describe a maximum version of their capacitated covering problem, $P2'$, in which k facilities are to be selected with the aim of maximizing the total profit of the clients covered. The problem is capacitated in that each facility has a single fixed capacity, the total client demand it can serve in a feasible solution.

They impose several restrictions on the problem to obtain an optimal algorithm in P.

The first of these restrictions is unit demand on each client, whose definition matches our own in the context of CVC. The second is what they term the *non-nestedness assumption*, which is defined as follows. For a client u_j, let $f'_{i(j)}$ be the leftmost facility of u_j's client interval, and $f''_{i(j)}$ the rightmost facility of its client interval. The non-nestedness assumption states that for any two clients $u_j < u_k$, $f'_{i(j)} < f'_{i(k)}$ implies $f''_{i(j)} \leq f''_{i(k)}$.

The final restriction, their *monotonicity property*, involves sorting facilities by the leftmost client they can cover, breaking ties by the rightmost client they can cover. The monotonicity property says that if clients $u_j < u_k$ are covered by distinct facilities $f_{i(j)}$ and $f_{i(k)}$, it is always possible to have $f_{i(j)} \leq f_{i(k)}$ in an optimal solution.

In addition to the concept of client demand, $P2'$ considers the costs of opening facilities and satisfying individual units of client demand, as well as penalties incurred for not satisfying demand, that maximum CVC does not. As we will discuss later, the version of the non-nestedness assumption used in section 4.1 is slightly less permissive than that of [22]. Algorithm 5 is generalized in section 4.2, where we relax our version of the non-nestedness assumption.

4.1 The non-nested algorithm

Once Algorithm 5 is established, we will discuss its connections to $P2'$ in section 4.1.1.

We begin the development of Algorithm 5 by reducing the 1-dimensional maximum uniform CVC problem to a capacitated covering problem on at most k facilities.

Our 1-dimensional k-capacitated covering problem is identical to uniform maximum CVC with an added constraint: we are to decide the optimal set of capacitated facility/-client assignments from among a set of facilities whose locations and ranges (and therefore capacities) are fixed on the line, in the same way clients are.

Lemma 4.1. *Suppose we are given a* 1*-dimensional uniform maximum CVC problem with client set* $\mathcal{C}$*. We can give a k-capacitated covering problem with facility set* $\mathcal{F}$ *and client set* $\mathcal{C}$*, such that* $|\mathcal{F}| = k \cdot |\mathcal{C}|$ *and that the greatest number of clients coverable with k facilities in* *the fixed facility (k-capacitated) setting coincides with that of the variable facility (uniform maximum CVC) setting.*

Proof. For each client $c \in \mathcal{C}$, place k facilities, one for each range/capacity pairing, such that the left end point of each facility's range coincides with the location of c on the location. Each facility is placed in the fashion of that depicted in Figure 4.1. This results in $k \cdot |\mathcal{C}|$

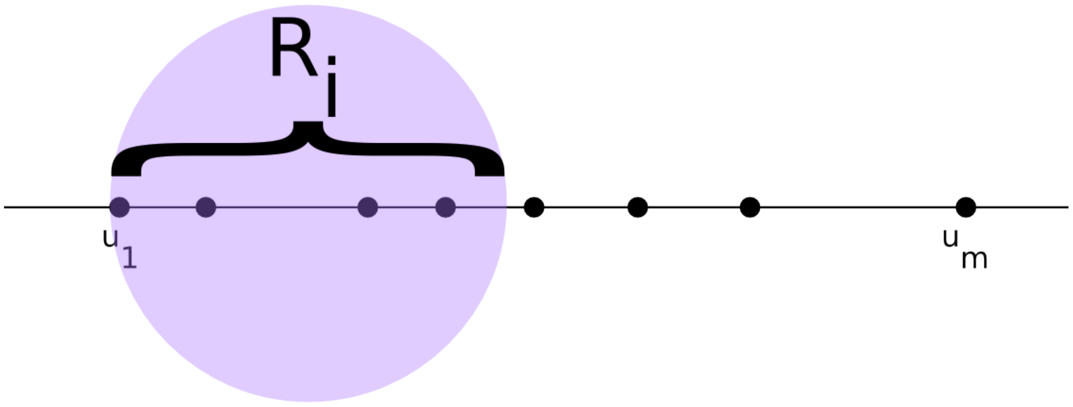

Figure 4.1: A facility of fixed capacity and range opened at the leftmost end point of a client.

fixed facilities on the line.

Now fix a solution $\mathcal{S}$ of the uniform maximum CVC problem. For each facility f, opened in S, designate lc_f as the leftmost client assigned to that facility, where f has range r_f. In the placement of fixed facilities, we know there is a facility f' of the same range and capacity as f whose left end point coincides with lc_f. It is apparent from the definition of f' that the range of f' includes all the clients assigned to f in S. Therefore, we can assign all clients of f in S to f' in the k-capacitated solution without violating f''s range and capacity constraints, which establishes the second claim of the lemma. $\square$

Let $\mathcal{F} \subseteq \mathbb{R}$ be the set of facilities and $\mathcal{C} \subseteq \mathbb{R}$ the set of clients. Each facility f has associated to it an interval of coverage referred to as R_f, where $|R_f| = r_f$. A facility can *cover* a client c under an assignment if $c \in R_f$.

Let $S : \mathcal{C} \to \mathcal{F}_\omega$, where $F_\omega = F \cup \{\omega\}$, be the assignment function and ω denote "nothing", a non-facility. Then $S(c) = \omega$ signifies that client c is unassigned.

Suppose that S satisfies the condition that for each $f \in \mathcal{F}$ and $c \in \mathcal{C}$, $c \in R_{S(c)}$ and $|S^{-1}(f)| \leq c_f$, where c_f denotes the capacity of f (S^{-1} is the inverse image of $f \in \mathcal{F}_\omega$, from the usual set-theoretic convention). We adopt the convention that $c \in R_\omega$ for all $c \in \mathcal{C}$, and that $c_\omega = +\infty$. Additionally, if up to k facilities $f \in \mathcal{F}$ satisfy $|S^{-1}(f)| > 0$, then S is a solution of the *k-capacitated variable covering problem* defined by the tuple $< \mathcal{C}, \mathcal{F} >$.

We number the clients in ascending order from left to right as $c_1, \ldots, c_m$, $m = |\mathcal{C}|$, $n = |\mathcal{F}|$. We refer to S as a *contiguous solution* if for all $f \in \mathcal{F}$, $S^{-1}(f) = [c_l(f), c_u(f)] \cap \mathcal{C}$, where $c_l(f) \leq c_u(f)$ are the first and last clients covered by $\mathcal{F}$ respectively. We allow $c_l(f) = c_u(f) = -\infty$ in order to signify $S^{-1}(f) = \emptyset$.

We order the facilities f in ascending order by $f + r_f/2$, and break ties by $f - r_f/2$. If two facilities tie under this scheme (their positions and ranges coincide), they are ordered arbitrarily. This defines a total ordering of $\mathcal{F}$.

Borrowing from the terminology of [22], we say the problem $< \mathcal{C}, \mathcal{F} >$ is *non-nested* if for every pair of distinct facilities $f_1, f_2 \in \mathcal{F}$, we have $R_{f_1} \not\subseteq R_{f_2}$ and $R_{f_2} \not\subseteq R_{f_1}$. Two nested facilities are shown in Figure 4.2.

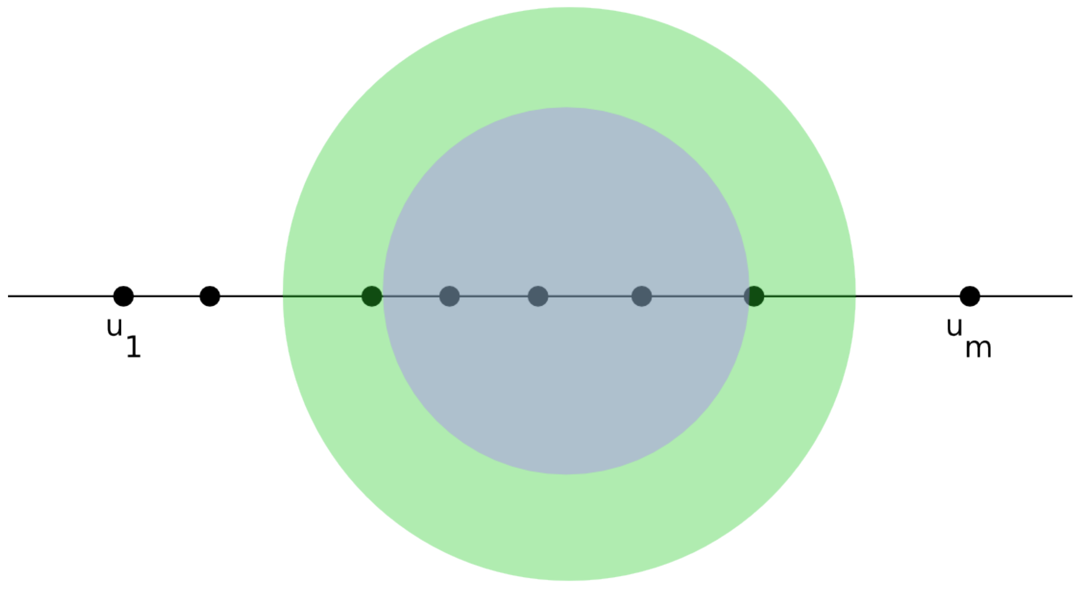

Figure 4.2: Two nested facilities.

Lemma 4.2. *For every solution $S : \mathcal{C} \to \mathcal{F}_\omega$ of a non-nested k-capacitated covering prob-*

lem, there is a contiguous solution S' covering the same number of clients.

Proof. Let S be a solution of a non-nested problem $P = < \mathcal{C}, \mathcal{F} >$. We can transform S into a contiguous solution using the following algorithm.

Start from the rightmost facility f assigned in S, and work leftwards on the clients $c \in R_f$. We suppose that the rightmost covered client is assigned to f. If this doesn't hold, find the rightmost assigned client, and the rightmost client assigned to f, and swap them. Since the number of clients covered has not decreased, take this new cover to be S.

Once the rightmost client assigned to f is found, go to its neighboring left client. If it's assigned to f, select it in place of the rightmost client. Otherwise, continue going left in this manner until a new client assigned to f is found. Between this client and the one selected previously, there is either an unassigned client, or a client assigned to a facility distinct from f.

In Figure 4.3, the process starts at client u_j, and continues leftward and it reaches the black client.

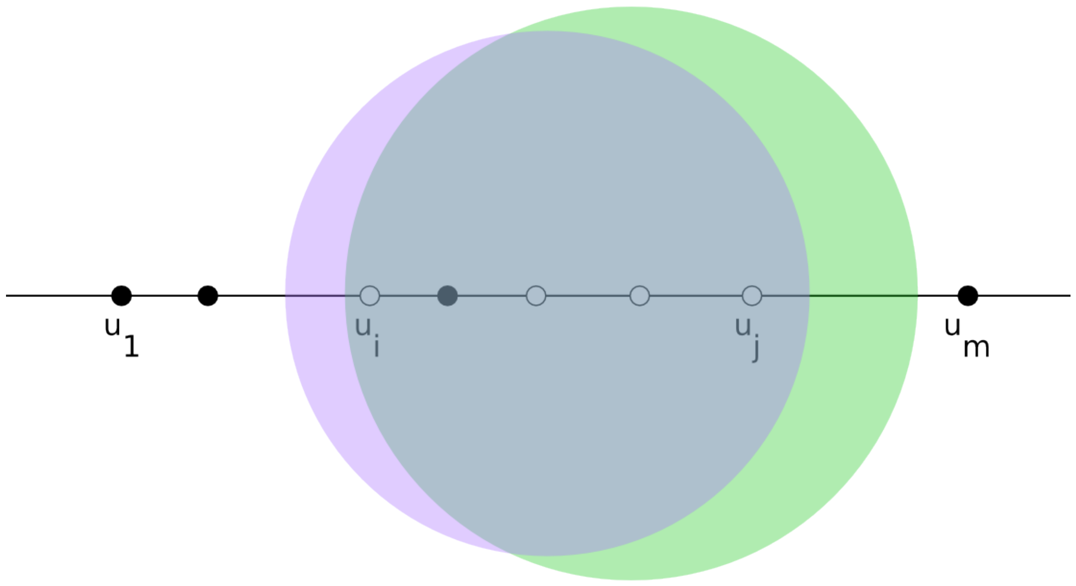

Figure 4.3: A non-contiguous solution where the white clients belong to the later (green) facility and the only black client within range is assigned to the purple facility.

In either case, we swap facility assignments to make the revealed client assignment of

f contiguous. This is depicted in Figure 4.4.

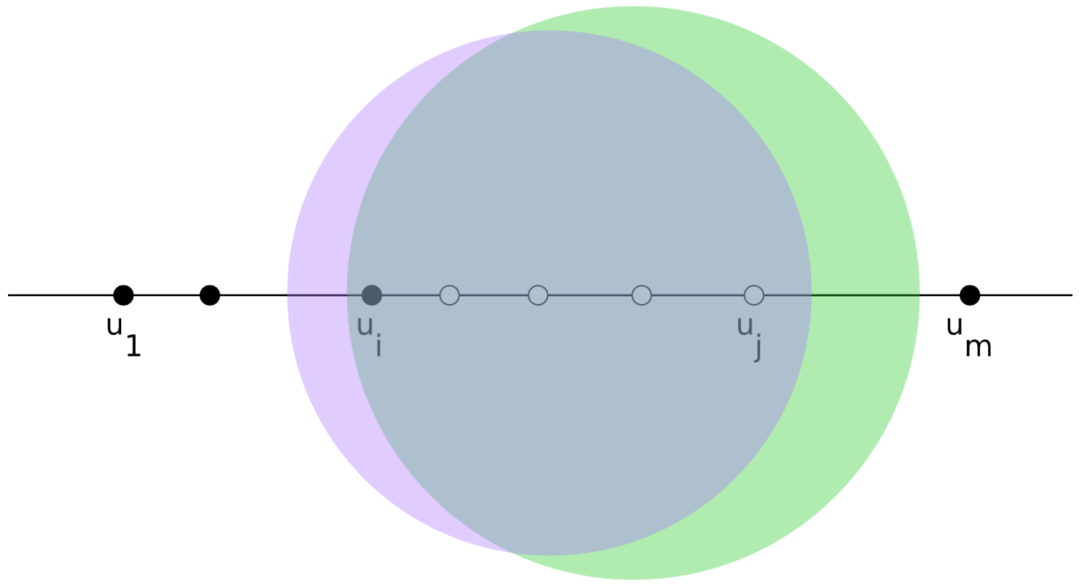

Figure 4.4: The assignments are swapped, so that the green facility's clients are contiguously located.

We continue until we've reached the leftmost $c \in R_f$. At that point, the clients assigned to f form a contiguous sequence in the client set. Furthermore, the number of clients assigned to any facility did not decrease from the original assignment of S, and the maximal client assigned to each facility in the total ordering did not increase.

Since the argument in the case of $|\mathcal{F}| = 1$ is trivial, we establish the statement of the lemma by induction. We use the last observation in the preceding paragraph to tie the induction hypothesis together with the argument on the clients of f. $\square$

By Lemma 4.2, every non-nested problem has a contiguous solution as an optimal solution. To motivate the complexity of the general algorithm, which computes optimal solutions for all 1-dimensional k-capacitated problems, we give an algorithm that computes optimal contiguous solutions.

We sort the facilities according to the above right endpoint / left endpoint ordering, and compute a dynamic program in k rounds. We use the optimal solutions for the r-capacitated covering problems on the first i facilities and c clients as our subproblems.

We define dp$[i][c][n][r]$ as the greatest number of clients $1 \leq c' \leq c$ covered in a contiguous solution of no more than r of the first i facilities, i being the last facility used in the covering and with exactly n clients assigned to i, supposing that $(c-n,c] \subseteq R_i$. If $(c-n,c] \not\subseteq R_i$, then dp$[i][c][n][r]$ is set to 0, as no covering as described by the table indices is possible.

An example of a solution corresponding to a dp entry in which $n = 2$ is depicted in Figure 4.5. In the facility ordering, the purple facility (facility i) comes last.

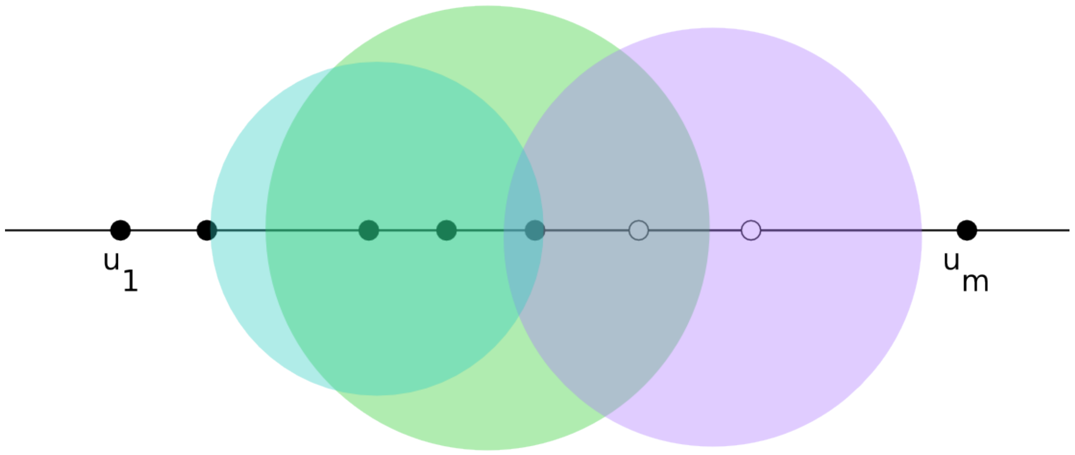

Figure 4.5: The white clients are assigned to the purple facility.

Once the purple facility's clients are assigned, it is stripped away and we consider the assignments to be made against the remaining facilities. The subproblem we get by decomposing from 4.5 produces the image in Figure 4.6.

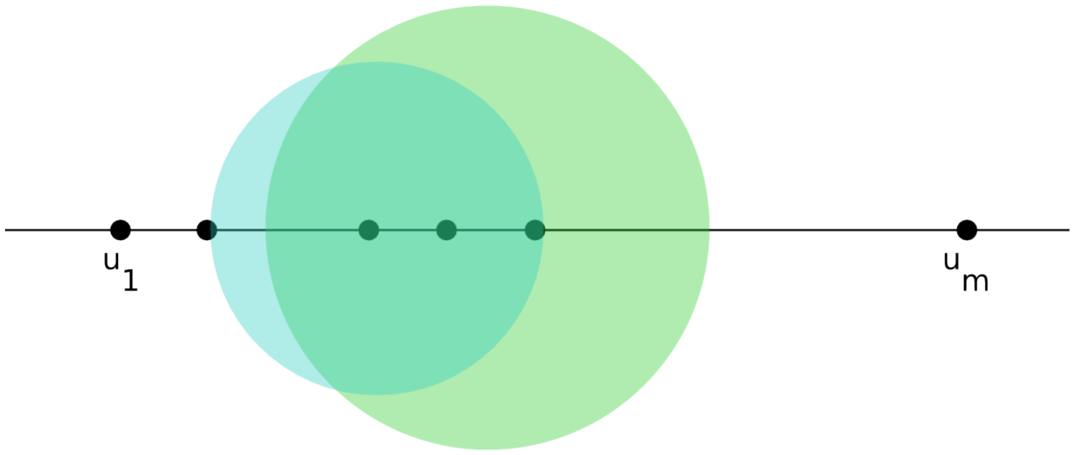

Figure 4.6: The purple facility of Figure 4.5 is stripped away.

With the relevant information stored in dp, m$[i][c][r]$ is set to the greatest number of clients $c' \in [1,c]$ coverable in a contiguous solution of no more than r of the first i facilities.

The tables are computed as in Algorithm 5, which is proved correct by induction on k.

Algorithm 5 Compute the optimal contiguous solution for a k-capacitated covering problem.

```
Require: $< \mathcal{C}, \mathcal{F} >$ is a chained problem.
 1: for $r$ from 0 to $k$ do
 2:     for $i$ from 0 to $|\mathcal{F}|$ do
 3:         for $c$ from 0 to $|\mathcal{C}|$ do
 4:             $\mathrm{m}[i][c][r] \leftarrow 0$
 5: for $r$ from 1 to $k$ do
 6:     for $i$ from 1 to $|\mathcal{F}|$ do
 7:         for $c$ from 1 to $|\mathcal{C}|$ do
 8:             for $n$ from 1 to $c$ do
 9:                 $\mathrm{dp}[i][c][n][r] \leftarrow 0$
10:             for $n$ from 1 to $\min(c_{f_i}, c)$ do
11:                 if $(c-n, c] \subseteq R_{f_i}$ then
12:                     $\mathrm{dp}[i][c][n][r] \leftarrow n + \mathrm{m}[i-1][c-n][r-1]$
13:             $\mathrm{m}[i][c][r] \leftarrow \max\{\mathrm{m}[i-1][c][r], \max_{1 \le n \le \min(c_{f_i}, c)} \mathrm{dp}[i][c][n][r]\}$
    return $\mathrm{m}[|\mathcal{F}|][|\mathcal{C}|][k]$
```

Proposition 4.3. *Algorithm 5 computes the size of the best contiguous solution of any k-capacitated covering problem (nested or non-nested) in $O(k|\mathcal{F}||\mathcal{C}|^2)$ time.*

Proof. In the $k = 1$ case, the correctness of dp is assured by greedily assigning clients within the range of facility i while respecting the capacity of facility i. Namely, the only non-trivial assignments to dp occur in line 12, which renders as

$$dp[i][c][n][1] \leftarrow n + 0$$

under the constraints $k = 1$, $(c-n, c] \subseteq R_{f_i}$, $n \le \min(c_{f_i}, c)$. Since we can only assign clients to a single facility, the optimal solution is obtained by selecting as many clients as the capacity and range of the selected facility will allow. Since all facilities are considered in turn, it's clear that dp (and subsequently m) optimize for the facility which can contain the most clients. Since the selected clients are contiguous by the definition of dp and its indices, we have shown the algorithm to compute the best contiguous solution in the case $k = 1$ without supposing the problem to be nested or non-nested.

For the induction step, suppose $k = r + 1$, and that each entry of the table slices $\mathrm{dp}[\cdot][\cdot][\cdot][r']$ and $\mathrm{m}[\cdot][\cdot][r']$ is the maximum number of clients coverable in the subproblem described by its indices for all $r' \leq r$.

Here the relation of $\mathrm{dp}[i][c][n][r]$ to previous subproblems is as follows. If each client of $(c-n, c]$ is assigned to facility i, then having selected i as the last of up to r facilities, we are left with the subproblem of assigning as many as we can of the first $c - n$ clients to up to $r - 1$ facilities, the last of which does not surpass facility $i - 1$ in the total ordering.

By induction, $\mathrm{m}[i-1][c][k-1]$ gives the greatest possible assignment of the clients of $[c_1, c]$ to up to $k-1$ of the first $i-1$ facilities, while respecting each of the assigned facilities' range and capacity constraints. Let $n \in \mathcal{Z}$ satisfy $(c-n, c] \subseteq R_{f_i}$ and $n \leq c_{f_i}$, the capacity of facility i. Then the n clients of $(c-n, c]$ are assignable to i without violating its range or capacity constriants. We add n facilities (implicitly assigned to i – there is no need to explicitly record this fact) to the maximum number of clients coverable in the subproblem described by the indices of $\mathrm{m}[i-1][c][r-1]$. If either $n > \min(c_{f_i}, c)$ or $(c-n, c] \subseteq R_{f_i}$, $\mathrm{dp}[i][c][n][r]$ retains its initial value of 0.

By the maximality of $\mathrm{m}[i-1][c][r-1]$, it follows that $\mathrm{dp}[i][c][n][r]$ is maximal also, if the adjoining assignment to i is feasible. $\mathrm{dp}[i][c][n][r] = 0$ if and only if the assignment of the clients of $(c-n, c]$ to i is infeasible.

As for $\mathrm{m}[i][c][r]$, the maximal assignment described by its indices assumes one of two forms. Either the i facility is not among the r or fewer facilities opened, in which case $\mathrm{m}[i][c][r] = m[i-1][c][r]$, or it is. If facility i is included, then by the previous argument, for some integer n, $\mathrm{dp}[i][c][n][r]$ describes the best contiguous solution possible by the previous argument, with n bound in the range $1 \leq n \leq \min(c_{f_i}, c)$.

Therefore $\mathrm{m}[i][c][r]$ is the number of clients covered in the best contiguous solution of the first c clients using up to r facilities, with none of the selected facilities surpassing i.

For the run time analysis, we multiply the limits of the four main loops together to obtain a run time of $O(k|\mathcal{F}||\mathcal{C}|^2)$. We note that the range check $(c-n,c] \subseteq R_{f_i}$ can be completed in constant time, using a static range query array that is computed exactly once. It takes $O(|\mathcal{F}|+|\mathcal{C}|)$ time to fill the array. □

4.1.1 Relation to Mirchandani *et al.*'s Problem $P2'$

In [22], problem $P2$ is defined as the following profit maximization problem:

"Select up to k facilities that maximize the total profit from serving some or all (non-nested) customers on the straight line."

Again, in problem $P2$ clients are located on the line and have pre-defined intervals of facilities that are able to cover them, up to some capacity. Client demands are uniform, and there are opening costs for facilities, as well as per-unit costs for satisfying client demand.

The non-nested k-capacitated covering problem of section 4.1 is a restrained version of $P2'$, with opening and per-unit costs set to 0. This can be seen through the following reduction. Fix an instance of the non-nested q-capacitated covering problem. We sort the facilities in the order used in [22], by the leftmost client covered, breaking ties by the rightmost client covered.

If we fix any client, the set of facilities containing that client is a contiguous interval in the sorted sequence of facilities. This follows from the non-nestedness of the k-capacitated problem from which we started. The set is exactly the customer interval of the fixed client in [22]. It is clear that no two customer intervals are nested, since that would imply the existence of facilities with nested ranges in our version of the problem.

The dynamic programming table $v(i,j,k,t)$ of Algorithm $DP2$ in [22] is structured almost identically to dp$[i][c][n][r]$: i (resp. i) determines the last facility opened, $1,\ldots,j$ $(1,\ldots,c)$ are the clients that can be covered, exactly k (n) clients are covered by i, and t (r) facilities are opened. The dynamic progrma of $DP2$ is considerably simpler, considering only single additions of clients to *i* at a time. The more complex algorithm presented in section 4.2 is to set the stage for the development of Algorithm 5, in which the non-nestedness assumption is abandoned completely.

4.2 The nested algorithm

We find that as we look to solving non-nested *k*-capacitated covering problems, the best contiguous solution isn't necessarily the optimal solution. Consider the problem shown in Figure 4.7. The circles represent facility ranges and the x's represent clients.

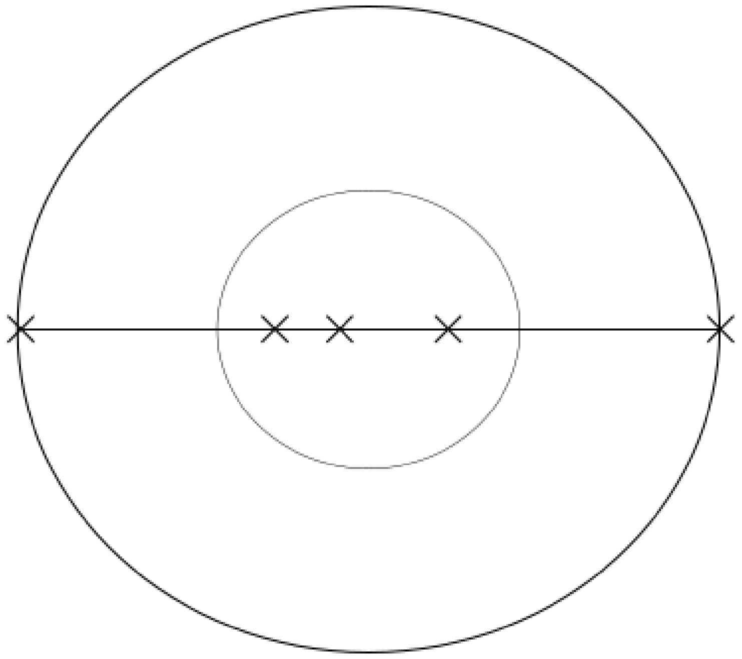

Figure 4.7: A 2-capacitated covering problem whose optimal solution is not a contiguous solution.

In Figure 4.7, the innermost facility has capacity 3 and the outermost facility has capacity 2. Plainly all five clients (the crosses) can be covered, by assigning the inner clients to the inner facility and the outer clients to the outer facility, but this cannot be arranged as a contiguous solution.

We generalize Algorithm 5 by giving special consideration to what we term interior facilities. That is, we compute optimal solutions of *k*-capacitated covering problems by memoizing the solutions of *interior subproblems*.

An interior subproblem is an r-capacitated problem restricted to a set of facilities whose ranges are entirely interior to a given facility's range. When selecting a facility as the last facility used in a covering, as dp does, we modify Algorithm 5 to range over i's interior subproblems. The selected interior solution will be augmented to contain i's client assignments, and the assignments of any clients to facilities preceding but not interior to i will be handled as before using dp.

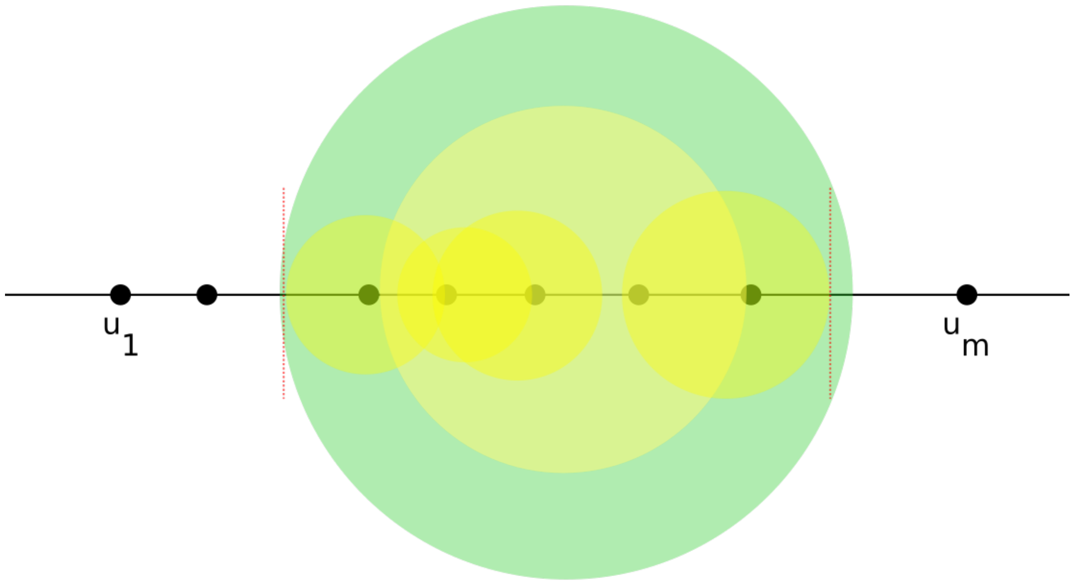

Figure 4.8: An interior subproblem.

The clients selected for assignment by subproblems will be stored in descending order in a sorted linked list. We will assign clients to i by adding the missing clients in order to the linked list, copying only those nodes that are surrounded by the added clients in the client ordering. This means that in any round of list augmentation, the number of nodes copied is $O(|\mathcal{C}|)$, and the original subproblem list on which the interpolation was based is not disturbed.

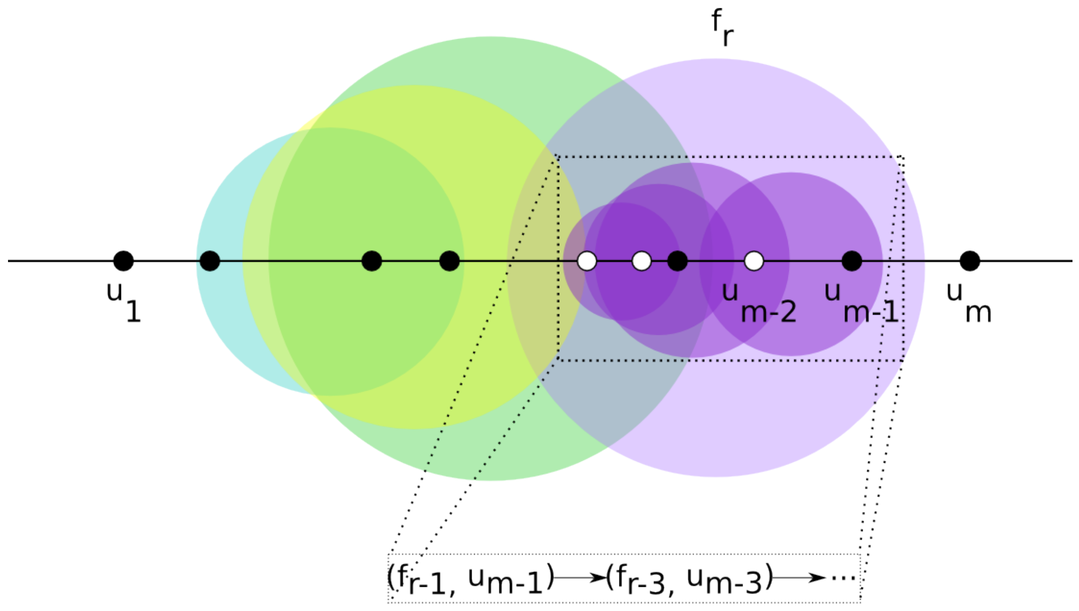

Figure 4.9: An interior subproblem with solution list indexed somewhere in sub.

The subproblem table has five dimensions. The first (c_l) is the leftmost client of the interior subproblem, the second (c_u) is the rightmost client, the third (f_l) is the leftmost facility of the interior subproblem and the fourth (f_u) is the right facility (again, not necessarily used). The fifth (p) gives the number of facilities used in the interior subproblem. Finally, sub$[c_l][c_u][f_l][f_u][p]$ is the maximum number of clients coverable by p facilities in the p-capacitated covering problem described by the indices. lst$[c_l][c_u][f_l][f_u][p]$ contains the linked list representation of the subproblem solution as described above.

$\mathcal{F}_{[f_l,f_u]}$ is the set of facilities whose right and left endpoints fall between f_l and f_u. It can be computed in $O(|\mathcal{F}|)$ time by traversing the sorted sequence of facilities.

The m table is much the same as in Algorithm 5, but without the middle (client) dimension. m$[i][r]$ gives the cost of the optimal solution of the r-capacitated covering problem restricted to the first i facilities.

Algorithm 6 fills the sub table by calling the generalized algorithm routine, *solve*, on each interior subproblem. For *solve* to be correct, we require every entry of the table slice sub$[\cdot][\cdot][\cdot][\cdot][0]$ to be 0. For $r > 0$, we require initially that sub$[\cdot][\cdot][\cdot][\cdot][r] = -1$ to indicate that we haven't yet memoized the solutions of any of those subproblems.

Algorithm 6 Fill the sub table.

```
Require: $\text{sub}[\cdot][\cdot][\cdot][\cdot][r] = \begin{cases} 0 & \text{if } r = 0 \\ -1 & \text{otherwise} \end{cases}$
1: procedure POPULATE_SUBS($\mathcal{F}$, $\mathcal{C}$, $k$)
2:     for $r$ from 1 to $k-1$ do
3:         for $i$ from $f_1$ to $f_{|\mathcal{F}|}$ do
4:             for $c_u$ from $c_{min}(i)$ to $c_{max}(i)$ do
5:                 for $c_l$ from $c_{min}(i)$ to $c_u$ do
6:                     for $f_u$ from $f_{min}(i)$ to $f_{max}(i)$ do
7:                         for $f_l$ from $f_{min}(i)$ to $f_u$ do
8:                             if $|\mathcal{F}_{[f_l,f_u]}| \geq r$ and $\text{sub}[c_l][c_u][f_l][f_u][r] = -1$ then
9:                                 $(new\_cost, new\_list) \leftarrow \text{SOLVE}(\mathcal{F}_{[f_l,f_u]}, \mathcal{C}_{[c_l,c_u]}, r)$
10:                                $\text{lst}[c_l][c_u][f_l][f_u][r] \leftarrow new\_list$
11:                                $\text{sub}[c_l][c_u][f_l][f_u][r] \leftarrow new\_cost$
```

$f_{min}(i)$ and $f_{max}(i)$ denote the least and greatest facilities interior to facility i respectively, and similarly for $c_{min}(i)$ and $c_{max}(i)$ with respect to clients. With Algorithm 6 defined, we are ready to give *solve*.

Some needed definitions: greatest_non_overlapping(i) maps facility i to the greatest facility $j \leq i$ such that the ranges of j and i do not overlap. If all preceding facilities overlap i, greatest_non_overlapping(i) returns -1. The function non_interior_overlapping(i) maps i to the set of facilities $j \leq i$ which do overlap i, but are not interior to i.

round_up is a procedure that greedily augments the interior subproblem specified by its arguments. It extends the interior subproblem solution by assigning as many clients as possible to i, using the linked list representation described above and assumed by $\text{lst}[c_l][c_u][f_l][f_u][r]$. Each entry of lst is a pointer to the head node of a linked list, containing three fields: a facility f, a client it covers c, and the address of the next node in the list named *next*. round_up is given here as Algorithm 8.

We prove several needed properties of round_up in Lemma 4.4.

Lemma 4.4. *Suppose the sorted linked list* $\text{lst}[c_l][c_u][f_l][f_u][p]$ *is a representation of the optimal solution of the subproblem corresponding to its indices described in the call, round_up$(i, c_l, c_u, f_l, f_u, p_i)$. Then Algorithm 8 returns a tuple of two values, such that*

Algorithm 7 Compute the optimal solution to a given k-capacitated covering problem.

```
1:  procedure SOLVE($\mathcal{F}$, $\mathcal{C}$, $k$)
2:      for $r$ from 1 to $k$ do
3:          for $i$ from 1 to $|\mathcal{F}|$ do
4:              $\mathrm{m}[i][r] \leftarrow m[i-1][r]$
5:              for $c$ from 1 to $|\mathcal{C}|$ do
6:                  for $n$ from 1 to $\min(c_{f_i}, c)$ do
7:                      if $(c-n, c] \subseteq R_{f_i}$ then
8:                          for $r_i$ from 1 to $i-1$ do
9:                              $\mathrm{dp}[i][c][f_{r_i}][r] \leftarrow 0$
10:                             $\mathrm{ml}[i][c][f_{r_i}][r] \leftarrow null$
11:                             for $l_i$ from 1 to $r_i$ do
12:                                 for $p_i$ from 0 to $r-1$ do
13:                                     $(pc, pl) \leftarrow \textsc{round\_up}(i, c-n+1, c, f_{l_i}, f_{r_i}, p_i)$
14:                                     if $pc > dp[i][c][f_{r_i}][r]$ then
15:                                         $\mathrm{dp}[i][c][f_{r_i}][r] \leftarrow pc$
16:                                         $\mathrm{ml}[i][c][f_{r_i}][r] \leftarrow pl$
17:                                     for $j \in \mathrm{non\_interior\_overlapping}(f_i)$ do
18:                                         $l \leftarrow \textsc{append}(\mathrm{ml}[j][c-n][f_{l_i-1}][r-p_i-1], pl)$
19:                                         $pc' \leftarrow pc + \mathrm{dp}[j][c-n][f_{l_i-1}][r-p_i-1]$
20:                                         if $pc' > \mathrm{dp}[i][c][f_{r_i}][r]$ then
21:                                             $\mathrm{dp}[i][c][f_{r_i}][r] \leftarrow pc'$
22:                                             $\mathrm{ml}[i][c][f_{r_i}][r] \leftarrow l$
23:                                     $i' \leftarrow \mathrm{greatest\_non\_overlapping}(i)$
24:                                     if $i' > -1$ then
25:                                         $l' \leftarrow \textsc{append}(\mathrm{top\_lvl\_lst}[i'][r-p_i-1], pl)$
26:                                         if $\mathrm{m}[i'][r-p_i+1] + pc > m[i][r]$ then
27:                                             $\mathrm{dp}[i][c][f_{r_i}][r] \leftarrow \mathrm{m}[i'][r-p_i+1] + pc$
28:                                             $\mathrm{ml}[i][c][f_{r_i}][r] \leftarrow l'$
29:                                     if $\mathrm{dp}[i][c][f_{r_i}][r] > \mathrm{m}[i][r]$ then
30:                                         $\mathrm{m}[i][r] \leftarrow \mathrm{dp}[i][c][f_{r_i}][r]$
31:                                         $\mathrm{top\_lvl\_lst}[i][r] \leftarrow \mathrm{ml}[i][c][f_{r_i}][r]$
        return $(\mathrm{m}[n][k], \mathrm{top\_lvl\_lst}[n][k])$
```

Algorithm 8 Augment an existing interior solution according to the provided bounds.

```
Require: $f_l \le f_u$
 1: function ROUND_UP($i$, $c_l$, $c_u$, $f_l$, $f_u$, $p$)
 2:     $n \leftarrow c_u - c_l + 1$
 3:     $scov \leftarrow 0$
 4:     $new\_list \leftarrow node(0, 0, null)$                ▷ A dummy node storing the list head
 5:     if $f_l \ge f_{\min(i)}$ then
 6:         $scov \leftarrow \mathrm{sub}[c_l][c_u][f_l][f_u][p]$
 7:         $assn \leftarrow \mathrm{lst}[c_l][c_u][f_l][f_u][p]$
 8:         $remaining\_clients \leftarrow \min(scov + c_{f_i}, n)$
 9:         $tail \leftarrow new\_list$
10:         $num\_added \leftarrow 0$
11:         while $assn \ne null$ do
12:             while $remaining\_clients + c - n > assn.c$ do
13:                 if $num\_added \ge \min(c_{f_i}, n - scov)$ then break
14:                 $tail.next \leftarrow new\ node(f_i, remaining\_clients + c - n, null)$
15:                 $tail \leftarrow tail.next$
16:                 $remaining\_clients \leftarrow remaining\_clients - 1$
17:                 $num\_added \leftarrow num\_added + 1$
18:             $tail.next \leftarrow$ copy of $assn$
19:             $tail \leftarrow tail.next$
20:             $remaining\_clients \leftarrow remaining\_clients - 1$
21:             $assn \leftarrow assn.next$
22:         while $remaining\_clients > 0$ do
23:             $tail.next \leftarrow new\ node(f_i, remaining\_clients + c - n, null)$
24:             $tail \leftarrow tail.next$
25:             $remaining\_clients \leftarrow remaining\_clients - 1$
        return $(\min(scov + c_{f_i}, n), new\_list.next)$
```

- *the first value is a linked list of facility/client pairs sorted in descending order of clients, containing copies of each of the nodes of* $\text{lst}[c_l][c_u][f_l][f_u][p]$*, the "old nodes." The new nodes describe client assignments to facility i, and the number of new nodes does not exceed the capacity of* f_i*, and*

- *the second value is the length of the list comprising the first value.*

Proof. We suppose $f_l \leq f_{min}(i)$ and establish some preliminary facts.

First, it is immediately apparent that $n \leq scov$, the subproblem solution described by $\text{lst}[c_l][c_u][f_l][f_u][p]$ is limited to the range of n clients, $[c_l, c_u]$, and $scov$ is its length, by assumption. Second, we observe that everywhere in its pseudocode, round_up decrements the variable *remaining_clients* whenever a node is added to the tail of *new_list*, which it eventually returns. round_up terminates only when *remaining_clients* is 0; therefore, the initial value of *remaining_clients* is the length of *new_list* at the time *round_up* terminates. The length of *new_list* is either $scov + c_{f_i}$, the length of $\text{lst}[c_l][c_u][f_l][f_u][p]$ plus the capacity of facility i, or n, the number of clients in the range $[c_l, c_u]$, whichever is smaller. Whatever the minimum value, it is clear that the length is large enough to accommodate every node of $\text{lst}[c_l][c_u][f_l][f_u][p]$, and that the number of new clients to be assigned to facility i does not exceed the capacity of i.

round_up begins by iterating down the list at $\text{lst}[c_l][c_u][f_l][f_u][p]$ one node at a time, aggressively plugging the gaps between clients belonging to adjacent nodes. The gaps are filled with assignments of clients to facility i, decrementing *remaining_clients* with each new assignment. This occurs whenever the candidate client $remaining\_clients + c - n$ is greater than the client at the lower bound $assn.c$. Since $assn.c \geq c_l$, the new client assignment cannot fall outside the range $[c_l, c_u]$. If during the execution of the inner loop, we find that *num_added* exceeds $\min(c_{f_i}, n - scov)$, we terminate the loop. $\min(c_{f_i}, n - scov)$ is exactly the number of new nodes that *new_list* will contain once round_up terminates, and so, once the limit of new nodes is reached, no new nodes are added. Then *remaining_clients* is exactly the length of the tail of $\text{lst}[c_l][c_u][f_l][f_u][p]$ that starts at *assn*, and we append that tail to *new_list*, until *assn* is *null*. The loop terminates, and the entry condition of the second while loop is not met, so *new_list* is returned along with its length in a tuple.

If the break condition on *new_added* is never incurred before *assn* becomes *null*, the inner loop is exited once $remaining\_clients + c - n > assn.c$ fails. There it fails because $remaining\_clients + c - n = assn.c$, since *remaining_clients* decreases by one unit at a time. The node at *assn* is copied, and the copy is appended to the intermediate list at *new_list*. *remaining_clients* is decremented, and construction of *new_list* resumes in this manner until *assn* is null, meaning that every node of lst$[c_l][c_u][f_l][f_u][p]$ has been copied and recorded in *new_list*.

The clients that remain to be assigned to *new_list* are greedily assigned to facility i in descending order. At this stage, every client in the range $[c_l, remaining\_clients + c - n] = [c - n, remaining\_clients + c - n]$ is unassigned. Further, we have $\min(c_{f_i}, n - scov) - num\_added = remaining\_clients$, where we know $\min(c_{f_i}, n - scov) - num\_added$ to be the number of new nodes yet to be added. Therefore, all new client assignments fall within the range $[c_l, c_u]$, and both claims made in the statement of the lemma are established. □

Finally, a generalized version of Lemma 4.2 allows us to claim that the optimized solutions produced by Algorithm 7 coincide exactly with the optimal solutions of the given k-capacitated covering problem. It uses the concept of the generalized contiguous solution, given here.

Definition 4.5 (Generalized contiguous solution)**.** Let $S : \mathcal{C} \to \mathcal{F}_\omega$ be an assignment of clients to facilities. S is a generalized contiguous solution if for every pair of facilities f_i, f_j such that $R_{f_i} \not\subseteq R_{f_j}$ and $R_{f_j} \not\subseteq R_{f_i}$ and $f_i < f_j$ in the facility ordering, we have $\max\{c : S(c) = f_i\} < \min\{c : S(c) = f_j\}$.

Lemma 4.6. *For every solution $S : \mathcal{C} \to \mathcal{F}_\omega$ of a k-capacitated covering problem, there is a generalized contiguous solution S' covering the same number of clients.*

The proof of Lemma 4.6 is virtually identical to that of Lemma 4.2. We summarize the top-level solve routine as follows.

Algorithm 9 Populate the sub table and solve the top level problem.

```
1: function TOP_LEVEL_SOLVE($\mathcal{F}$, $\mathcal{C}$, $k$)
2:     POPULATE_SUB($\mathcal{F}$, $\mathcal{C}$, $k$)
3:     return SOLVE($\mathcal{F}$, $\mathcal{C}$, $k$)
```

Proposition 4.7. *Algorithm 9 computes the optimal generalized contiguous solution of any k-capacitated covering problem in $O(k^3|\mathcal{F}|^7|\mathcal{C}|^5)$ space and time.*

Proof. To prove Algorithm 9 correct, we perform strong induction on k.

In the base case, we have $k = 1$. We want to show dp$[i][c][f_{r_i}][1]$ contains the greatest number of clients coverable in the subproblem indicated by the loop indices, and similarly for the list representation of the solution contained in ml$[i][c][f_{r_i}][1]$.

First, populate_subs is called with $k = 1$, and since the outer loop variable r is less than k, it terminates immediately, having done nothing.

solve is called next. We fix the loop variables i, c and n such that $(c-n, c] \subseteq R_{f_i}$. Let the loop variables l_i, r_i take any value in their allowed ranges. We note that $r = 1$ and $p_i = 0$ unconditionally.

round_up is called in line 13, with arguments instantiated to the values of various loop variables, the last of which is $p = 0$. Therefore, in the internals of round_up, $scov = 0$ and $assn = null$, the empty list. $remaining\_clients = \min(c_{f_i}, n)$, where n is the number of clients in the range $[c_l, c_u]$. Since $assn = null$, round_up skips down to the second while loop, where new_list is augmented to contained the entire range of clients in $(c-n, remaining\_clients + c - n]$, all of them assigned to facility i.

After round_up returns, we go to the next line, where we begin looping over the facilities whose ranges overlap the range of i but are not contained within it. l is set to the list described in the previous paragraph, since ml$[j][c-n][f_{l_i-1}][r-p_i-1]$ is $null$. pc is then the number of clients assigned in l. Throughout the loop, dp$[i][c][f_{r_i}][1]$ is maximized over the largest assignment of clients to i, with ml$[i][c][f_{r_i}][1]$ set to the list describing it.

Once the loop terminates, the outcome of greatest_non_overlapping(i) is immaterial, since $k = 1$ and we cannot add to the assignments of clients to i by round_up any further. Thus, by appealing to the correctness of round_up through Lemma 4.4, we have proved the proposition for the base case. From this, it is apparent that m$[i][1]$ contains the greatest number of clients coverable by a single facility in the facility range $[f_1, f_i]$, and that top_lvl_lst$[i][1]$ contains its linked list representation.

Now for $k > 1$, suppose the proposition holds for every configuration $< \mathcal{F}, \mathcal{C} >$ whenever $k' < k$. By induction, populate_subs places the correct values into the arrays sub and lst, since their final indices, those deciding the number of facilities placed in the corresponding coverings, are limited to values less than k.

Similarly, when we enter *solve*, for every $r < k$ we can suppose that every entry of the tables dp, ml, top_lvl_lst, and m all hold the defined values whenever their final indices are equal to r. This also holds under the inductive hypothesis.

Now let i, c, n, l_i and p_i be loop variables such that $(c - n, c] \subseteq R_{f_i}$ holds. Suppose also that the best possible solution of the subproblem described by the indices of dp$[i][c][f_{r_i}][k]$ can be realized under the constraints imposed by this particular combination of n, l_i and p_i.

By Lemma 4.4 and the induction hypothesis, the result of line 13 returns a covering, pl, and the number of clients it covers, pc. The covering returned is an optimal covering of at most p_i facilities, selected from among the facility range $[f_{l_i}, f_{r_i}]$ internal to f_i and f_i itself, that covers clients in the range $[c_l, c_u]$. If $pc >$ dp$[i][c][f_{r_i}][k]$, the cost pc and corresponding solution pl are written to the corresponding entries of the intermediary cost and list arrays, in case they represent the best solution of the subproblem.

Otherwise, in the solution whose representation will be written to the table entry ml$[i][c][f_{r_i}][k]$ when the inner loop terminates, the assignment returned by round_up will be extended by some placement of $k - p_i - 1$ facilities to the left of, and non-interior to, i. The extension is described at dp$[j][c-n][f_{l_i - 1}][k - p_i - 1]$ for some facility j non-interior to i.

If facility j overlaps i, then that value of j is instantiated in the inner loop. Then the final value of $\text{ml}[i][c][f_{r_i}][k]$ is written to l, and the total cost of the extension and the assignment of clients to i and its interior facilities is pc'. Since this solution is best in the presumed case, its size and representation are propagated to $\text{dp}[i][c][f_{r_i}][k]$ and $\text{ml}[i][c][f_{r_i}][k]$ respectively.

If facility j is disjoint from i, it lies to the left of the facility greatest_non_overlapping(i), which is to say the condition $i' > -1$ is satisfied in the pseudocode. Then the value of the best possible extension is given at m[greatest_non_overlapping(i)]$[r - p_i - 1]$, with representation top_lvl_lst[greatest_non_overlapping(i)]$[r - p_i - 1]$, and $\text{dp}[i][c][f_{r_i}][k]$ and $\text{ml}[i][c][f_{r_i}][k]$.

From this, it becomes clear that as i varies, top_lvl_lst$[i][k]$ and $\text{m}[i][k]$, after their final assignments, are the optimal values as defined according to their indices.

With the induction hypothesis proved for all $k \geq 1$, it follows that solve returns the number of clients covered in the optimal placement of k facilities and the representation of that placement.

For the cost analysis, we begin by analyzing the run time of solve. Multiplying the lengths of the loops together, we get $O(k^2|\mathcal{F}|^3|\mathcal{C}|^2)$ iterations of the inner body of code beginning at line 13. The run time of the inner body of code is bounded by time $O(|\mathcal{C}| + |\mathcal{F}| \cdot |\mathcal{C}| + |\mathcal{C}|)$. round_up takes $O(|\mathcal{C}|)$ time to run, and the loop on the elements of non_interior_overlapping(i) takes $O(|\mathcal{F}| \cdot |\mathcal{C}|)$ time. The last step, taken if $i' > 0$, takes time $O(|\mathcal{C}|)$, the length of top_lvl_lst$[i'][r - p_i - 1]$. Therefore, solve runs in time $O(k^2|\mathcal{F}|^4|\mathcal{C}|^3)$.

The rest of the execution of top_level_solve is spent in the body of populate_subs. Multiplying the lengths of its loops together produces $O(k|\mathcal{F}|^3|\mathcal{C}|^2)$ iterations. Combined with the runtime of solve, and the range tree query used to compute $\mathcal{F}_{[f_l, f_u]}$, we get an overall run time of $O(k^3|\mathcal{F}|^7|\mathcal{C}|^5)$. □

Theorem 4.8. *Suppose we have a 1-dimensional uniform maximum CVC problem with client set $\mathcal{C}$ and P range/capacity pairs. Then the problem can be solved optimally in $O(k^3P^7|\mathcal{C}|^{12})$ time, where k is the number of facilities opened.*

Proof. By Lemma 4.1, every maximum CVC problem with uniform demand can be transformed into an equivalent k-capacitated covering problem, where k is the number of facilities to be opened. The two problems are equivalent in the sense that they share the same maximum number of coverable clients drawn from the same client set.

Furthermore, by Lemma 4.6, the optimal solution of any k-capacitated covering problem can be expected to take the form of a generalized contiguous solution. Proposition 9 says that the best generalized batch assignment of any k-capacitated covering problem on $|\mathcal{C}|$ clients and $|\mathcal{F}|$ facilities can be computed in $O(k^3|\mathcal{F}|^7|\mathcal{C}|^5)$ time and space.

The value of $|\mathcal{F}|$ in the k-capacitated problem obtained from the algorithm of Lemma 4.1 is $P \cdot |\mathcal{C}|$. Tying these observations together, we have that any 1-dimensional maximum uniform CVC problem can be solved in time and space $O(k^3P^7|\mathcal{C}|^{12})$. $\square$

Chapter 5

Disjoint Weighted Barrier Coverage

Wireless sensor networks are used to provide detection services over an area of interest. Sensors are placed on the boundary, or perimeter, of the area. The perimeter to be covered is the *barrier*. If the sensors are placed such that any perimeter intrusion is sure to be detected, the sensors are said to cover the barrier. The covering of the perimeter by the sensors is also referred to as a *barrier coverage*, as the sensors act as a barrier to intruders.

In one-dimensional coverage problems, the barrier is represented by a horizontal line segment and sensors are initially placed on the line containing the line segment. The goal is to compute new positions for some subset of the given sensors, ensuring that every point in the barrier segment is within the detection range of some sensor. In typical two dimensional coverage problems, sensors are represented as points in the plane and assigned a radius of coverage while the barrier is represented as the boundary of some closed planar region.

In this chapter we consider the problem of *One-Dimensional Disjoint Barrier Coverage* on a line segment. Sensors s_i are represented as finite intervals of $\mathcal{R}$, with center points x_i and detection radii r_i. Each sensor s_i triggers an alarm upon detecting movement at any point in the interval $[x_i - r_i, x_i + r_i]$.

Barriers are represented by the interval $[0, L]$, for some real $L > 0$. The family of problems is termed *disjoint* because sensors are placed in initial positions where their detection ranges do not meet the barrier, so that $[x_i - r_i, x_i + r_i] \cap [0, L] = \emptyset$ for each sensor s_i.

We evaluate the quality of a covering in terms of the MinSum problem, which is concerned with minimizing the total distance travelled by all sensors to their final positions in the solution. In MinSum, each sensor is shifted m_i units to the right or left, depending on whether $x_i < 0$ or $x_i > L$, respectively, to align with the furtherest uncovered point on the barrier.

This leaves us with the goal of minimizing the sum $\sum_{s_i \in S} w_i \cdot m_i$, where $w_i > 0$ is a real-valued weight assigned to the sensor s_i. The inclusion of a weight allows the mathematical model of sensor placement to more accurately reflect certain practical realities. In addition to allowing sensors to have non-uniform detection ranges, some of the sensors used may have suffered the effects of wear, deterioration and obsolescence and so, bear higher mobilization costs.

After showing that unweighted, one-sided disjoint barrier coverage is NP-hard, we will develop a 2-approximation algorithm for the one-sided disjoint weighted case. It will then be used to derive an FPTAS for the two-sided disjoint weighted case.

5.1 NP-hardness of the one-sided, disjoint case

In order to motivate the development of approximation algorithms, we show that the MinSum barrier coverage problem is NP-complete by reducing from the Partition problem, which is defined in the following way. Given a sequence of positive integers $a_1, \ldots, a_n$ and an integer B such that $\sum_{1 \le i \le n} a_i = 2B$, find a subset of integers $\mathcal{S}$ such that $\sum_{a_i \in \mathcal{S}} a_i = B$.

From the instance $(\{a_i\}_{1 \le i \le n}, B)$, we create a barrier from the line segment $[0, B]$, and for each integer $a_i > 0$, we create a sensor with range $r_i = a_i/2$ and distance $d_i = |x_i| - r_i \ge 0$ from 0, $x_i < 0$, for x_i determined later in the reduction. Let S be any subset of sensors whose total range covers all of the barrier. Using the formula for the cheapest movement covering the barrier completely, we have

$$c(S) = |S| \cdot B - \sum_{i=1}^{|S|} (|S| - i) \cdot a_{s_i} + \sum_{i=1}^{|S|} d_{s_i} \tag{5.1}$$

where the sequence $\{s_i\}_{1 \le i \le |S|}$ indexes the intervals of S, and is again ordered so that $r_{s_1} \ge$

$r_{s_2} \geq \ldots \geq r_{s_{|S|}}$. Similarly, $l(S)$ denotes the total length of the sensors in S, so that

$$l(S) = \sum_{s \in S} l_s$$

where l_s is the length of sensor s.

We choose d_i in order to ensure that, for any covering sensor subsets S and S', $l(S) > l(S')$ implies $c(S) > c(S')$. An optimal algorithm for *DisjointMinSum* then gives an optimal algorithm for the Partition problem. We need only apply the optimal algorithm for *Disjoint-MinSum* to the reduction of the Partition problem instance and check that the solution has total length B.

To that end, we decide the values of d_i. Suppose S, S' are covering subsets of sensors satisfying $l(S) > l(S')$. Using the cost formula, we get

$$\begin{aligned} c(S) - c(S') &= (|S| - |S'|) \cdot B - \sum_{i=1}^{|S|} (|S| - i) \cdot a_{s_i} \\ &\quad + \sum_{i=1}^{|S'|} (|S'| - i) \cdot a_{s'_i} + \sum_{i=1}^{|S|} d_{s_i} - \sum_{i=1}^{|S'|} d_{s'_i} \\ &= B \sum_{i=1}^{|S|} \left(d^B_{s_i} + 1 - \frac{|S| - i}{B} \cdot a_{s_i} \right) \\ &\quad - B \sum_{j=1}^{|S'|} \left(d^B_{s'_j} + 1 - \frac{|S'| - j}{B} \cdot a_{s'_j} \right) \end{aligned}$$

where we define $d^B_i = d_i / B$. Then $c(S) - c(S') > 0$ is equivalent to

$$\sum_{i=1}^{|S|} \left(d^B_{s_i} + 1 - \frac{|S| - i}{B} \cdot a_{s_i} \right) > \sum_{j=1}^{|S'|} \left(d^B_{s'_j} + 1 - \frac{|S'| - j}{B} \cdot a_{s'_j} \right)$$

Let $d_i^B = (2B+1)\cdot a_i - 1 > 0$. In particular, we have

$$\begin{aligned}\sum_{i=1}^{|S|} d_{s_i}^B + 1 - \frac{|S|-i}{B}\cdot a_{s_i} &= \sum_{i=1}^{|S|}(2B+1-\frac{|S|-i}{B})\cdot a_{s_i} \\ &> \sum_{i=1}^{|S|} 2B\cdot a_{s_i}\end{aligned}$$

where the inequality holds by the following argument. $|S|-i < |S| \leq B$, and so $1-(|S|-i)/B > 0$. With the assumptions $a_{s_1} > 0$ and $|S| \geq 1$, we establish strict inequality.

From $l(S)-l(S') \geq 1$, we get that

$$\sum_{i=1}^{|S|} 2B\cdot a_{s_i} - \sum_{j=1}^{|S'|} 2B\cdot a_{s'_j} = 2B\cdot(l(S)-l(S')) \geq 2B$$

It is easy to see that

$$2B \geq \sum_{j=1}^{|S'|}\left(1-\frac{|S'|-j}{B}\right)\cdot a_{s'_j}$$

giving $c(S) > c(S')$, combined with the earlier inequalities.

5.2 A 2-approximation algorithm for the one-sided, disjoint weighted case

We begin by describing a 2-approximation algorithm for a version of the weighted barrier coverage under MinSum using the initial restriction that sensors lie only to the left of the barrier. The algorithm is given as pseudocode in Figure 10.

The algorithm is greedy, filling the line by packing sensors from the right. It selects one sensor in each step, the witness to the minimum of the cost function $cost(s,x) := w_s(d_s + x)/\min(l_s,x)$ as it ranges over $s_i \in S$, with x fixed as the length of the line segment that remains to be covered. The loop iterates until a full covering is produced.

We show that Algorithm 10 is a 2-approximation to the weighted, left-lying disjoint MinSum problem. The idea is to fix two solutions to the problem instance, the first any

Algorithm 10 Calculate a covering of the line.

```
Require: $\sum_{s_i \in S} l_i \geq L$
1: $Covering \leftarrow \emptyset$
2: $x \leftarrow L$
3: while $x > 0$ do
4:     $s \leftarrow \arg\min_{s_i \in S}\{w_i(d_i + x)/\min(l_i, x)\}$
5:     $S \leftarrow S - \{s\}$
6:     $x \leftarrow x - l_s$
7:     $Covering \leftarrow Covering \cup \{s\}$
   return $Covering$
```

covering of least cost, and the second the covering given by Algorithm 10.

The bounds proof is structured as a procedure whose subroutines are given as lemmas handling the various cases. The approximate sensor solution is iterated over one sensor at a time, starting from the rightmost sensor and moving leftward. The cost of each sensor in the approximate solution is bounded by no more than twice the cost of sensors in the optimal solution. We refer to the rightmost end point of the remaining approximate sensors as the cursor. That is, the variable x assumed the value of the cursor in line 6 of Algorithm 10 at a unique time in its execution. We therefore refer to the "cursor at x", and consider x to denote a "time" in the execution of Algorithm 10.

Along the way, we derive invariants that must be maintained for the assumptions made by the lemmas to hold. This way, the lemmas can be applied on a case-driven basis as required, since the conditions of at least one of the lemmas are guaranteed to hold with each change of the cursor.

The first invariant describes what we term the *viability* of sensors in the optimal solution with respect to the approximate sensor at the cursor. That is, when sensor s_i was selected by Algorithm 10, it was the minimum of the function $s \rightarrow cost(s, x)$ over the set of candidate sensors for the cursor x. We also see that Algorithm 10 denotes the set of candidate sensors as S.

Let S_x be the value of the set variable S in Algorithm 10 at time x, and let app refer to the set of sensors comprising the solution given by Algorithm 10. Let opt be a fixed optimal solution. An optimal sensor $t \in S_x$ is said to be *viable* with respect to an approximate sensor s at x if $cost(s,x) \leq cost(t,x)$.

Viability is summarized in Definition 5.1. A sensor $t \in$ opt is *viable* at x if $t \in S_x$, implying $cost(s,x) \leq cost(t,x)$.

Definition 5.1 (Viability)**.** Let $s \in$ app be the sensor at cursor x, so that s witnesses the minimum of the function $s \rightarrow cost(s,x)$ over S_x. If $t \in \text{opt} \cap S_x$, then t is viable with respect to s at x.

Let $p_{s,T}$ be the location of the rightmost end point of sensor s in solution $T \in \{\text{app}, \text{opt}\}$. If $s \notin T$, we set $p_{s,T} = -\infty$. Define the *ratio* of sensor s as the ratio of its weight to length, denoted as $ratio(s) = w_s/l_s$.

An important property used throughout the bounds procedure is the *Order Preservation Property*, which states that the sensors comprising any optimal solution are ordered in descending order of ratio.

Lemma 5.2 (Order Preservation Property)**.** *Let opt be an optimal solution, with sensors s, $s' \in opt$ such that $p_{s,opt} \leq p_{s',opt}$. Then*

$$ratio(s) \geq ratio(s')$$

Proof. By transitivity, we may suppose without loss of generality that s and s' are packed adjacently in opt, with $p_{s,\text{opt}} \leq p_{s',\text{opt}}$. Suppose for the sake of contradiction that

$$\frac{w_s}{l_s} < \frac{w_{s'}}{l_{s'}}$$

This means that exchanging the positions of s and s' in opt yields a change in cost

$$w_s l_{s'} - w_{s'} l_s < 0$$

This is a contradiction. □

In the course of developing the bounds procedure, it will sometimes be necessary to split optimal solution sensors into one or more sensors, and to distribute the cost of the original among its constituent parts. More specifically, let $s \in$ opt and let the number $l_{s'}$ satisfy $0 < l_{s'} < l_s$. Starting from the left end point of s, designate a contiguous length of s as sensor s' of length $l_{s'}$, weight $w_{s'} = (l_{s'}/l_s)w_s$ and distance $d_{s'} = d_s$. Then set $l_{s''} = l_s - l_{s'}$, $w_{s''} = (l_{s''}/l_s)w_s$, and $d_{s''} = d_s$. Then s has been split into sensors s' and s'', both of which have been given length, weight and distance attributes. We refer to the procedure as fractioning.

A useful fact is that the ratio and cost of each of the split sensors do not differ from those of the original.

Lemma 5.3 (Invariance of Cost and Ratio under Fractioning)**.** *Suppose that sensor s has been fractioned into sensors s' and s''. Then $cost(s,x) = cost(s',x)$ and $ratio(s) = ratio(s')$ for all $x > l_s$, and similarly for sensors s and s''.*

Proof. Expanding the definitions of the weights, we note that

$$ratio(s') = w_{s'}/l_{s'} = (l_{s'}w_s/l_s)/l_{s'} = w_s/l_s = ratio(s)$$

and similarly for s''.

Similarly, for the weighted movement to length ratio at any $x > l_s$, we have

$$cost(s',x) = w_{s'}(d_{s'}+x)/l_{s'} = w_s(d_s+x)/l_s = cost(s,x)$$

and the same follows for $cost(s'',x)$ by a similar argument. □

Next, we define and prove two facts used crucially in the bounds procedure.

Lemma 5.4. *Let $a_1,\ldots,a_n$, $b_1,\ldots,b_n$ be positive real numbers. Then*

$$\min_{1\le i\le n} \frac{a_i}{b_i} \le \frac{a_1+\ldots+a_n}{b_1+\ldots+b_n} \le \max_{1\le i\le n} \frac{a_i}{b_i}$$

Proof. We prove the second inequality, $(a_1 + \ldots + a_n)/(b_1 + \ldots b_n) \leq max_{1 \leq i \leq n} a_i/b_i$, by induction. It is vacuously true in the case of $n = 1$, so the base case occurs at $n = 2$.

Let $a, b, c, d > 0$ and suppose without loss of generality that $a/b \leq c/d$. The assertion $(a+c)/(b+d) > c/d$ is equivalent to $(a+c)d > c(b+d)$, while $a/b \leq c/d$ is equivalent to $ad \leq bc$. The two statements contradict each other, so for $a/b \leq c/d$ to hold, $(a+c)/(b+d) \leq c/d$ must be true.

Now let $n = k+1$ for $k > 2$ and assume the righthand equality of the lemma holds for $n = k$. Let $A_k = \sum_{i=1}^{k} a_i$ and $B_k = \sum_{i=1}^{k} b_i$.

If $A_k/B_k \leq a_{k+1}/b_{k+1}$, then we adapt the argument of the base case to show that $(A_k + a_{k+1})/(B_k + b_{k+1}) \leq a_{k+1}/b_{k+1} \leq max_{i=1,\ldots,k+1} a_i/b_i$.

If on the other hand $A_k/B_k > a_{k+1}/b_{k+1}$ is true, we use the base case argument to show that $(A_k + a_{k+1})/(B_k + b_{k+1}) \leq A_k/B_k$, and combining it with the statement of the induction hypothesis at k, we get $A_k/B_k \leq \max_{i=1,\ldots,k} a_i/b_i = \max_{i=1,\ldots,k+1} a_i/b_i$, the desired inequality.

To prove the lower bound, we swap the labels of a_i and b_i for every i, apply the righthand inequality just shown, and take the recripocals of either side. □

The second fact leverages the Order Preservation Property to induce upper bounds on the movement costs of sensors. Recall that $p_{s,\text{opt}}$ is the position of sensor s in opt, and similarly for $p_{s,\text{app}}$.

Lemma 5.5 (Shifting trick). *Let $s \in opt$, and for $1 \leq i \leq n$, $s_i \in opt$. Suppose $p_{s,opt} \leq p_{s_i,opt}$ for each $1 \leq i \leq n$ and $\sum_{i=1}^{n} l_{s_i} \leq l_s$. Then*

$$w_s \geq \sum_{i=1}^{n} w_{s_i}$$

Proof. From Lemma 5.2 we have $w_s/l_s \geq w_{s_i}/l_{s_i}$ for each $1 \leq i \leq n$. From the righthand inequality of Fact 5.4 we get

$$\frac{w_s}{l_s} \geq \max_{1\leq i\leq n} \frac{w_{s_i}}{l_{s_i}} \geq \frac{\sum_{i=1}^{n} w_{s_i}}{\sum_{i=1}^{n} l_{s_i}}$$

From $\sum_{i=1}^{n} l_{s_i} \leq l_s$, we may conclude that $w_s \geq \sum_{i=1}^{n} w_{s_i}$. □

At last, we are ready to give the first case of the bounds procedure in Proposition 5.6.

Proposition 5.6. *Suppose Invariant 5.1 holds and that $s \in opt$ is the sole element of opt, meaning that s is at the cursor x, and $l_s \geq x$. Then the total cost of sensors $s' \in app$ is no more than twice the cost of s.*

Proof. Let $s_n \in$ app be the sensor at cursor x and $s_1, \ldots, s_{n-1}$ the remaining sensors in app in order starting from the left. We assume that the left end of the barrier, point 0, is covered by sensor s_1.

Since s_1 is needed to cover the left end point, we must have $\sum_{i=2}^{n} l_{s_i} < x \leq l_s$. $l_s \geq x$ implies $s \neq s_i$ for each $2 \leq i \leq n$, and thus, $s \in S_{x-\sum_{i=j+1}^{n} l_{s_i}}$ for each $2 \leq j \leq n$, and so by Invariant 5.1 is viable with respect to each s_j at $x - \sum_{i=j+1}^{n} l_{s_i}$.

From this observation and Lemma 5.4, we derive

$$\frac{w_s(d_s+x)}{x} \geq \max_{2\leq i\leq n} \frac{w_{s_i}(d_{s_i}+x-\sum_{j=i+1}^{n} l_{s_j})}{l_{s_i}} \geq \frac{\sum_{i=2}^{n} w_{s_i}(d_{s_i}+x-\sum_{j=i+1}^{n} l_{s_j})}{\sum_{i=2}^{n} l_{s_i}}$$

Since $x \geq \sum_{i=2}^{n} l_{s_i}$, it is clear that $w_s(d_s+x) \geq \sum_{i=2}^{n} w_{s_i}(d_{s_i}+x-\sum_{j=i+1}^{n} l_{s_j})$.

Now, if $s = s_1$, then the cost of s_1 can be paid by debiting a second copy of the cost of s to the approximate solution. If $s \neq s_1$, then s is viable with respect to s_1 at $p_{s_1,\text{app}}$ and a similar bounding argument shows we can pay for s_1 using a second charge to the cost of s. □

We will assume from this point forward that $s \in$ opt at the cursor x has its full length contained within the barrier at $[0,L]$. Any exceptions to the assumption will be dispatched using Proposition 5.6. In effect, the equalities

$$cost(s,x) = \frac{w_s(d_s+x)}{\min(l_s,x)} = \frac{w_s(d_s+x)}{l_s}$$

are assumed to hold unconditionally for every sensor s and at every point $x \in [0,L]$ from now on.

For the next part of the bounds procedure, we consider the case where there is a single sensor in opt underlying the sensor at the cursor in app.

Proposition 5.7. *Suppose that Invariant 5.1 holds. Let $s' \in app$ and $s \in opt$ be the sensors at x, such that $s \in S_x$. Suppose that $x \geq l_{s'}$ and $l_s > l_{s'}$ as in Figure 5.1. Then we can pay the cost of s' using the fractional cost of s proportional to the length of s'. By contracting the unused part of s to a sensor $s'' \in opt$ with $cost(s'',x) = cost(s,x)$ and $ratio(s'') = ratio(s)$, we have that Invariant 5.1 is preserved under the assumption that $s'' \in S_{x-l_{s'}}$.*

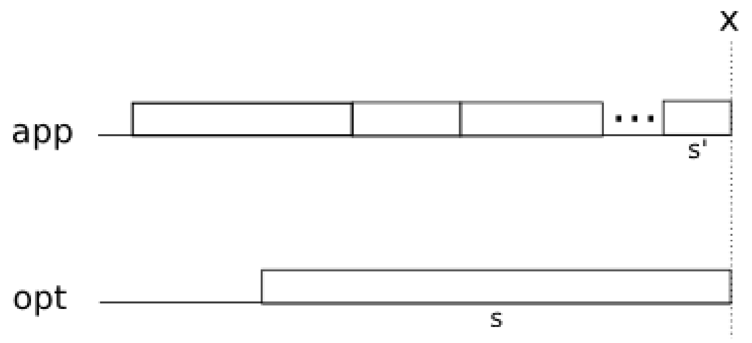

Figure 5.1: Bounding the cost of a sensor using a larger sensor underlying it in opt.

Proof. By Invariant 5.1, we have

$$w_{s'}(d_{s'}+x) \leq l_{s'}\frac{w_s(d_s+x)}{l_s} \leq w_s(d_s+x)$$

We move the cursor x to $x - l_{s'}$, having paid the cost of s' using the fraction of the cost of s that forms the right-hand side of the bound.

We fraction the unused part of s to a new sensor s'', where the right end point of s'' falls on the succeeding cursor $x - l_{s'}$ with $w_{s''} = (1 - l_{s'}/l_s)w_s$ and $l_{s''} = l_s - l_{s'}$. By Lemma 5.3, we have $cost(s'', y) = cost(s, y)$ and $ratio(s'') = ratio(s)$ for all $y > 0$.

We observe also that $s \neq s'$ and $s \in S_x$, and together these facts imply $s \in S_{x-l_{s'}}$. Therefore, we may substitute s'' for the remaining part of s in the succeeding cursor step and assert that $s'' \in S_{x-l_{s'}}$ without violating Invariant 5.1. $\square$

The other major case occurs when there are multiple optimal sensors underlying the sensor at the cursor in app.

Proposition 5.8. *Let s be the approximate sensor at cursor x, and suppose that Invariant 5.1 holds. Suppose we find by stabbing downward from the left endpoint of s a sensor distinct from that found in the optimal solution at x, as in Figure 5.2, with each sensor on and to the right of the stabline an element of S_x. Then the cost of the approximate sensor at x can be bounded by no more than twice the cost of the optimal sensors that lie on and to the right of the stabline.*

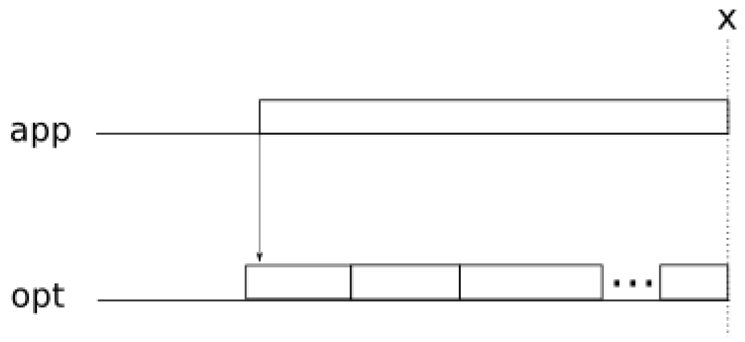

Figure 5.2: A single sensor with multiple sensors underlying it.

Proof. We label the sensors used in the optimal solution for ease of reference in Figure 5.3.

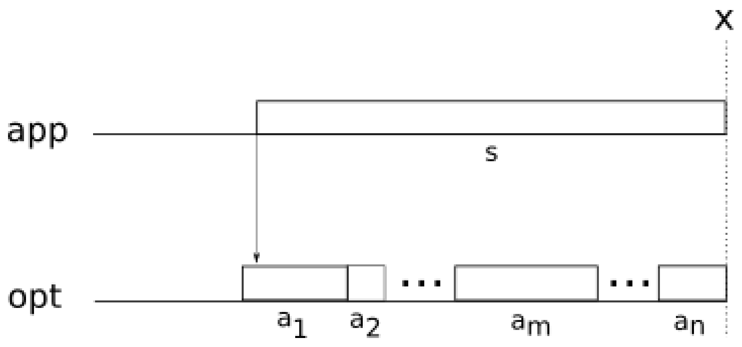

Figure 5.3: The underlying sensors are labelled.

As noted in the figure, s is bisected, fractioning the sensor in opt about the bisection line (below, a_m) if necessary. This process is depicted in Figure 5.4.

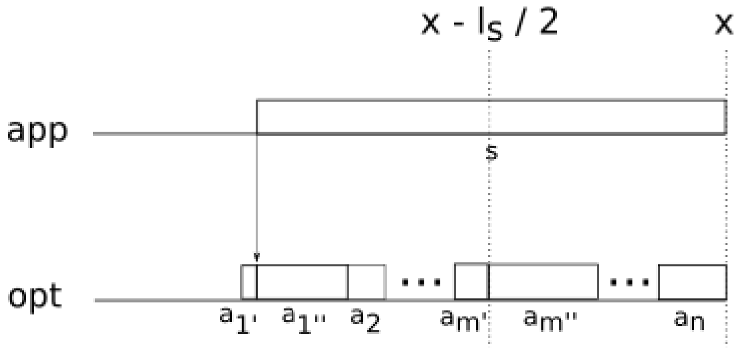

Figure 5.4: The bisected sensor is fractioned, if necessary.

The optimal sensor a_m is fractioned at the bisector, so that $w_{a'_m} = (l_{a'_m}/l_{a_m})w_{a_m}$ and $w_{a''_m} = (l_{a''_m}/l_{a_m})w_{a_m}$. By Lemma 5.3, $ratio(a_{m'}) = ratio(a_{m''}) = ratio(a_m)$ and $cost(a_{m'}, y) = cost(a_{m''}, y) = cost(a_m, y)$ for all $y > 0$.

Next, we compact the optimal sensors lying to the right of the bisector to right-align with x, as in Figure 5.5.

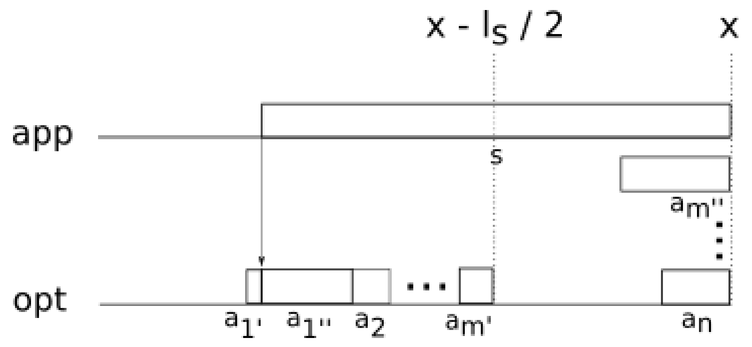

Figure 5.5: The sensors lying to the right of the bisector are compacted to *x*.

The extra charge needed to perform the compaction is not included in the costs tallied in the optimal solution. In order to pay for the compaction, we charge each sensor $a_{m''}, \ldots, a_n$ twice, and deduct the cost of the compaction from the secondary charge of each compacted sensor, as depicted below on the *sec* line. See Figure 5.6.

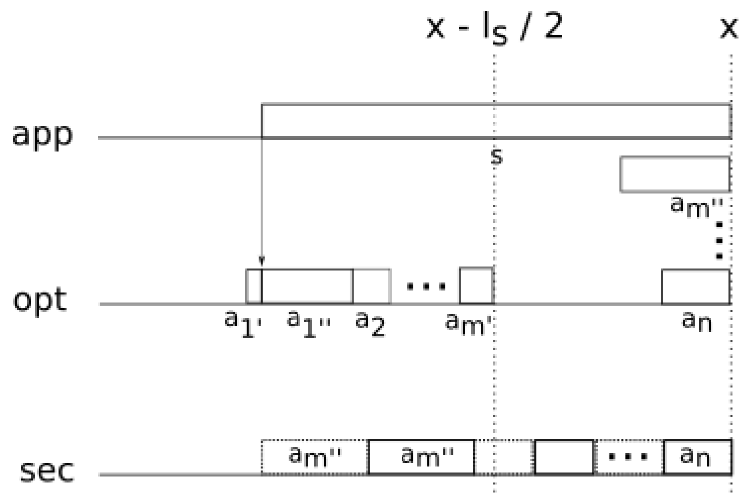

Figure 5.6: The residual charges of the compacted sensors are mapped on *sec*.

The dashed sensors in *sec* correspond to empty space of length equal to the length of the labelled sensor. Of the compacted sensors, $a_{m''}$ is furthest to the left, and the charge required to compact it is equal to the length of all the sensors following it in opt. In *sec*, $a_{m''}$ is deducted the same total length, retracted back from its original position at $p_{a_{m''},sec} = p_{a_{m''},\text{opt}}$. This is to compensate for the cost of the shift made to it in opt, since the goal is to charge $a_{m''}$ no more than twice its original cost. The same compensation scheme is applied to the shifting movements undertaken by each of the compacted sensors, leaving the solid sensors in *sec* to represent the unused portions of the secondary charges.

Since $a_{m''}, \ldots, a_n \in S_x$, we have by Fact 5.4

$$\frac{\sum_{i=m''}^{n} w_{a_i}(d_{a_i}+x)}{\sum_{i=m''}^{n} l_{a_i}} \geq \frac{w_s(d_s+x)}{l_s}$$

Because $\sum_{i=m''}^{n} l_{a_i} = l_s/2$, we have $\sum_{i=m''}^{n} w_{a_i}(d_{a_i}+x) \geq w_s(d_s+x)/2$.

To pay for the second half of $w_s(d_s+x)$, we use the untouched optimal sensors to the left of the bisector to shift the remaining secondary charges to x. That, we shift each of the solid sensors in *sec* to align with x by using the untapped charge of the sensors to the left of the bisector in opt, as follows.

Starting from $x - l_s/2$, we trawl leftward, selecting the sensors in the interval $I_{a_{m''}} = [x - l_s/2 - l_{a_{m''}}, x - l_s/2]$, fractionally if necessary; since both *ratio* and *cost* are invariant under fractioning, we can safely ignore the possibility of fractioning in the rest of the argument.

If sensor $a \in I_{a_{m''}}$, then by Lemma 5.2 we have $ratio(a) \geq ratio(a_{m''})$. Since $\sum_{a \in I_{a_{m''}}} l_a = l_{a_{m''}}$, we apply Lemma 5.5 to get $\sum_{a \in I_{a_{m''}}} w_a \geq w_{a_{m''}}$.

We note that for every point $p \in I_{a_{m''}}$, $p - (x - l_s) \geq l_s/2 - l_{a_{m''}}$. By charging every interval in $I_{a_{m''}}$ twice, we get $(l_s - 2l_{a_{m''}})(\sum_{a \in I_{a_{m''}}} w_a)$ in charge. By the inequality just obtained, we have

$$(l_s - 2l_{a_{m''}}) \sum_{a \in I_{a_{m''}}} w_a \geq (l_s - 2l_{a_{m''}}) w_{a_{m''}}$$

which is the minimum charge necessary to shift $a_{m''} \in sec$ to right-align with x.

The same argument serves to move the remaining sensors in opt to right align with x, with the lengths adjusted to match the distance of each sensor in *sec* from x. Once each copy of $a_{m''}, \ldots, a_n$ is aligned to x, we pay for the second half of $w_s(d_s+x)$ as before.

Lastly, we should note that $a_{1'}$ can be assumed to be an element of S_{x-l_s}. This follows from Lemma 5.3 and the assumptions made in the statement of this lemma. $\square$

We might be tempted to adapt the argument of Proposition 5.8 to an entire approximate solution by bisecting the whole of the optimal solution and applying the charging scheme. Since the proof of Proposition 5.8 depends on every optimal sensor in the range underlying s at cursor x being a member of S_x, the scheme cannot work in every case.

We could easily conceive of a weighted MinSum problem in which some sensor t satisfies $p_{t,\text{opt}} = p_{t,\text{app}} = L$, implying $t \notin S_x$ for any $x < t$. That is, the cost of t would be applicable as a bound exactly once, at $x = L$, rendering a second copy of t useless in the charging scheme of Proposition 5.8, where it plays an essential role.

Since the propositions we now have all assume that optimal sensors near x satisfy Invariant 5.1, the only cases left to consider are those in which the condition fails for optimal sensors near x. An optimal sensor $t \notin S_x$ if and only if t appeared in app after the approximate sensor s at x. Equivalently, t was drawn by the algorithm from some S_y where $y > x$, in preference to s.

Therefore, to get a fully general bounds procedure, we must give special consideration to sensors $t \in \text{app} \cap \text{opt}$. To that end, we ask whether charging t twice is enough to shift t to x.

If the answer is yes, we apply Proposition 5.9. If the answer is no, we apply Proposition 5.10.

Proposition 5.9. *Suppose the approximate sensor s at cursor x satisfies $2p_{s,opt} \geq x$. Then the cost of $s \in app$ can be paid entirely by twice the cost of $s \in opt$.*

Proof. We charge $s \in \text{opt}$ twice, noting

$$2w_s(d_s + p_{s,\text{opt}}) \geq w_s(d_s + 2p_{s,\text{opt}}) \geq w_s(d_s + x)$$

□

Proposition 5.10. *Suppose the approximate sensor s at cursor x satisfies $2p_{s,opt} < x$ and that the subinterval of sensor placements $[x - p_{s,opt}, x] \subseteq opt$ contains no gaps. Then the* *cost of $s \in app$ can be paid using the cost of $s \in opt$, and a length of l_s in sensors belonging to $[x - p_{s,opt}, x] \subseteq opt$.*

Proof. We break the proof into cases centered around the total size of the sensors in the set $I = \{t \in \text{app} : t \in [x - p_{s,\text{opt}}, x] \cap \text{opt}\}$.

We define $|I|_{size}$ to be the sum of the lengths of the sensors of I. If $|I|_{size} = \sum_{t \in I} l_t \geq l_s$, we select sensors $t \in I$ by priority of largest $p_{t,\text{opt}}$, until the total length of selected sensors is exactly l_s; we fraction one of the sensors to achieve this, if necessary. Let I' be this subset of I, so that $|I'|_{size} = l_s$.

Since $2p_{s,\text{opt}} < x$ and $t \in I$ implies $t \in [x - p_{s,\text{opt}}, x]$, we have $p_{s,\text{opt}} < p_{t,\text{opt}}$ for all $t \in I$. Therefore, $ratio(s) \geq ratio(t)$ for all $t \in I$ by Lemma 5.2.

From Fact 5.4, we have

$$\frac{w_s}{l_s} \geq \frac{\sum_{t \in I'} w_t}{\sum_{t \in I'} l_t} \geq \frac{\sum_{t \in I'} w_t}{l_s}$$

so that $w_s \geq \sum_{t \in I'} w_t$.

Since $|x - p_{t,\text{opt}}| \leq p_{s,\text{opt}}$ for each $t \in I'$, we can use a single charge of s to move each $t \in I'$ to right-align with x. We notice that each $t \in I$ is viable with respect to s at x, since t precedes s in app.

Since $|I'|_{size} = l_s$, we can pay for $s \in$ app using a single charge of $s \in$ opt and a single charge of each $t \in I'$, like so

$$w_s p_{s,opt} + \sum_{t \in I'} w_t (d_t + p_{t,opt}) \geq \sum_{t \in I'} w_t (d_t + x) \geq w_s (d_s + x)$$

To prove the first inequality, we exploit the fact that $p_{s,opt} + p_{t,opt} \geq x$ for every $t \in I'$ in conjunction with the earlier inequality $w_s \geq \sum_{t \in I'} w_t$. The second follows from the viability of every sensor of I' with respect to s at x alongside $\sum_{t \in I'} l_t = l_s$. The process is concluded by substituting a gap for $s \in$ opt and for each $t \in I'$. We note that this is an adaptation of the shifting trick first used in Proposition 5.8.

If $|I|_{size} < l_s$, we use the same shifting trick to move each $t \in I$ to right-align with x, charging the shift to $s \in$ opt. This leaves s with a fraction of $(l_s - |I|_{size})/l_s$ of a single charge remaining, which we use to shift $l_s - |I|_{size}$ in total length of sensors (any sensors) in $[x - p_{s,\text{opt}}, x] - I$ to x.

The sensors we select to shift, whether in I or not, are viable with respect to $s \in$ app at x. By definition, the selected sensors not in I were never present in app, and so are elements of S_x, making them viable with respect to s at x.

Once more, the cost of s is paid and the selected sensors moved by s and $s \in$ opt itself are replaced by gaps after the charge scheme is concluded. □

We have left open the problem of how to address the presence of gaps in opt. We especially want to guard against the possibility of gaps showing up near x, which could preclude the use of earlier propositions which depend on a sufficient length of optimal sensors being present near x.

To deal with gaps, we note that, in the statements of the previous two propositions, we charged the costs of t and each of the shifted sensors u in opt only once. We could instead charge the used optimal sensors twice, to get two copies of each of them shifted to x. As before, the first copy of each optimal sensor selected is used to pay for s while the second copy of each optimal sensor is cached for later use.

We introduce an invariant relating the total size of the gaps in opt and the total size of the sensors stored in the cache. The cache is implemented as a priority queue, PQ. The order of the sensors in the priority queue is the same we used to select the shifted sensors in the proof of Proposition 5.10, where sensors $t \in$ opt are stored in preference of largest $p_{t,app}$.

Definition 5.11. The total length of sensors in the priority queue PQ is equal to the total size of gaps in opt. Furthermore, for every $t \in PQ$ we have $p_{t,\text{app}} \leq x$ where x is the cursor.

Looking back on the proofs of Propositions 5.9 and 5.10, we see that the new invariant is maintained in either case. Proposition 5.9 closes the gaps it introduces without contributing to the cache, while Proposition 5.10 caches copies of the sensors whose lengths in opt are substituted by precisely the gaps introduced. Similarly, the earlier propositions create no gaps and cache no sensors.

Before we give the full bounds procedure, we introduce a final invariant describing a lower bound on the distance separating the nearest gap in opt from the cursor x.

Specifically, for every gap g in opt, we maintain a so-called "active region", a gapless interval of contiguous sensors r_g of size large enough that the shifting trick always has an adequately sized pool of sensors from which to draw. Every cached sensor t added to PQ after the formation of gap g will be tagged as belonging to g, which we denote as $tag(t) = g$.

Thus, every sensor in PQ is tagged as belonging to a unique gap g, and g is closed at the exact time the last sensor tagged g is removed from PQ.

Definition 5.12 (Total size of active region with respect to gaps)**.** Let g be a gap in opt. Then there is a contiguous (gapless) region r_g right-aligned at the cursor in opt, whose length is equal to the total size of sensors lying to the left of g in opt. Furthermore, at the time of any cursor x, the following statements hold:

- For any sensor $s \in r_g$, $p_{s,\text{app}} \leq p_g = \min\{p_{t,\text{app}} : t \in PQ, tag(t) = g\}$, and
- $\sum_{t \in PQ: tag(t)=g} l_t = l_g$, where l_g is the length of g.

Finally, in place of the notation opt, we use opt_C to denote a subset of opt in which some sensors have been replaced by gaps, shifted leftward from their original positions to close previous gaps, or retained their original positions in opt. We will use the notation app_C to refer to the current state of app, whose configuration of sensors experiences no change apart from when sensors are discarded at the cursor.

Proposition 5.13. *(Bounding procedure) Suppose we have sensor configurations app_C and opt_C satisfying the conditions of Invariants 5.1, 5.11 and 5.12. Then the total cost of sensors in app_C is no more than twice the total cost of the sensors in opt_C plus the cost of sensors in the priority queue.*

Proof. By induction on the number of sensors whose cost remains to be bounded in app_C.

Suppose $n = 1$. A single sensor is in app_C, with the possibility of sensors punctuated by gaps "below" it in opt_C, as depicted in Figure 5.7.

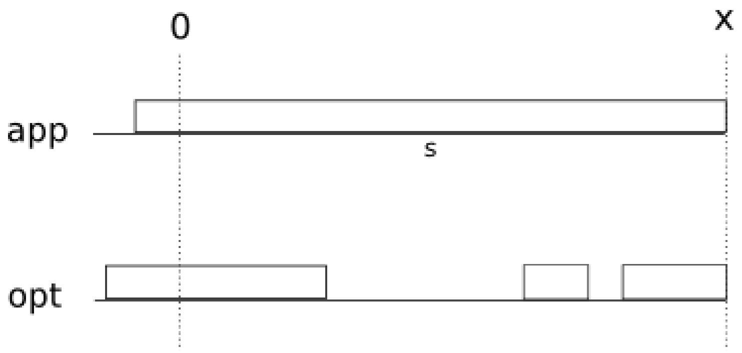

Figure 5.7: A single sensor in app overlying some gaps.

By Invariants 5.1 and 5.11, we can pay for exactly the proportion of s equal to the total length of the gaps using the cost of sensors drawn from the priority queue. If this does not cover the cost of s entirely, the configuration is made to resemble Figure 5.8, where s is fractioned to s', the remaining unpaid length of s, with the sensors in opt_C compacted to align with the new cursor p_x.

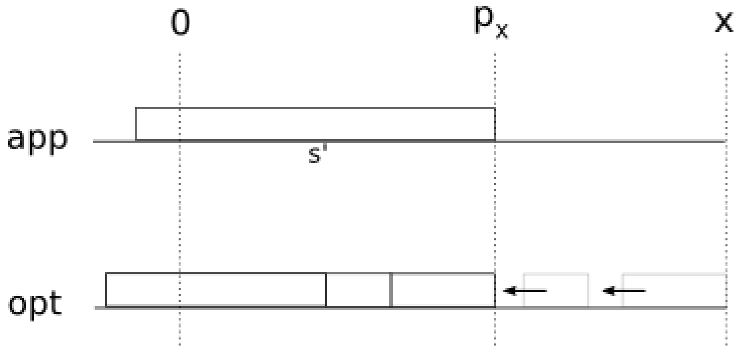

Figure 5.8: The gaps are closed by compact sensors to the left.

Since each $t \in \text{opt}_C$ is distinct from s (with the lone possibility of the sensor t containing 0, but this makes no difference to the argument), we have $t \in S_x$. By Invariant 5.1, the sensors $t \in \text{opt}_C$ satisfy

$$\frac{w_s(d_s + x)}{\min(x, l_s)} \leq \frac{w_t(d_t + x)}{\min(x, l_t)}$$

and therefore, since $p = p_x/x \leq 1$ is the unpaid portion of s,

$$\begin{aligned} p \cdot \frac{w_s(d_s + x)}{\min(x, l_s)} &\leq p \cdot \frac{w_t(d_t + x)}{\min(x, l_t)} \\ &\leq \frac{w_t(d_t + p \cdot x)}{\min(x, l_t)} \\ &\leq \frac{w_t(d_t + p_x)}{\min(x, l_t)} \end{aligned}$$

With the conditions of Propositions 5.6, 5.7 and 5.8 all satisfied, we conclude the bound with whichever of the three applies.

Now suppose $n = k + 1$ and that the statement of the proposition holds if app_C has k or fewer sensors. We consider the sensor s at cursor x by case.

Case 1: $s \in \text{app}_C \cap \text{opt}_C$

If $2p_{s,\text{opt}_C} \geq x$, we apply Proposition 5.9 to pay for $s \in \text{app}_C$, closing the gap left by s by compacting the optimal sensors to the right of s leftward. It is easy to see that Invariants 5.1 and 5.11 are maintained after the shift.

To show that Invariant 5.12 is maintained, suppose there is a gap g such that $p_{s,\text{opt}_C} \in r_g$ where r_g is the corresponding active region. Since Invariant 5.12 holds at time x by the induction hypothesis, we have that $p_{s,\text{app}_C} < p_{t,\text{app}_C}$ for all sensors t in the priority queue such that $tag(t) = g$. The inequality is strict because $p_{s,\text{app}_C} \geq 0$, and all non-negatively valued positions are unique.

Additionally, Invariant 5.12 tells us the length of sensors t in the priority queue tagged with gap g is equal to exactly the current length of g. By Invariant 5.11, $p_{t,\text{app}_C} \leq x$, but since sensor s is now right-aligned to x, we must have $x = p_{s,\text{app}_C} \geq p_{t,\text{app}_C}$ for every such t. This is a contradiction to the last sentence of the previous paragraph, since the positive length of gap g attests to the presence of sensors t with $tag(t) = g$ in the priority queue.

Therefore, $p_{s,\text{opt}_C} \notin r_g$. Since g was selected arbitrarily, Invariant 5.12 must remain true after Proposition 5.9 is applied, since s does not lie in any active region, and the gaps and priority queue are all left untouched.

If $2p_{s,\text{opt}_C} < x$, we apply Proposition 5.10 with a modification to its argument made to maintain Invariant 5.12. Let $u_1, \ldots u_n \in \text{opt}_C$ be the first l_s in total length of sensors u satisfying $2p_{u,\text{opt}_C} < p_{u,\text{app}_C}$ leftward from x, selected in preference of largest p_{u,opt_C}. We note that $p_{s,\text{opt}_c} \leq p_{u_i,\text{opt}_c}$ for every $i = 1, \ldots, n$ and that any sublength of s itself can be considered the last of these sensors if necessary, which guarantees that such a sequence of u's exists.

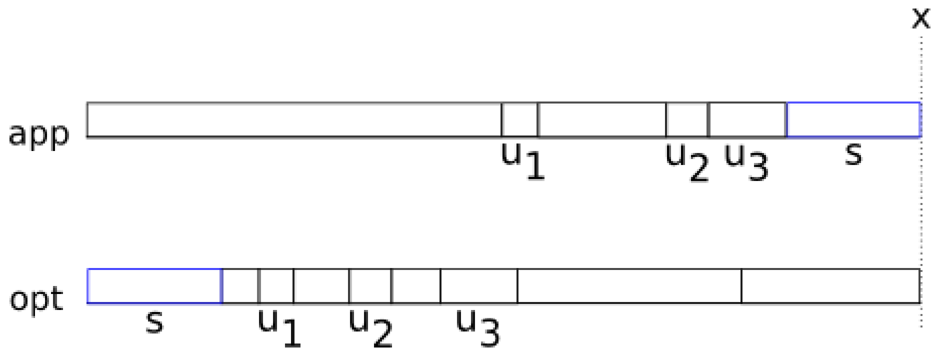

Figure 5.9: The modification in a case where $n = 3$.

We substitute the sensors u_i for s in providing the charge to shift the sensors selected by Proposition 5.10 in the range $[x - p_{s,\text{opt}_C}, x] \cap \text{opt}_C$ forward to x. That is, for each sensor u_i, we select a length l_{u_i} of sensors $t \in [x - p_{u_i,\text{opt}}, x] \cap \text{opt}_C$ in the manner described in Proposition 5.10, which are shifted to x. We do this in descending order, starting with the rightmost sensor in opt, u_n.

This means the active region r_{u_n} contains the active region $r_{u_{n-1}}$, which contains region $r_{u_{n-2}}$, and so on, all the way down to region r_{u_2} containing region r_{u_1}. As secondary copies of sensors are added to the priority queue, it is clear that Invariant 5.12 is maintained among the regions r_{u_i}. Supposing we modify Proposition 5.10 to follow this greedy approach whenever it is applied, Invariant 5.12 must hold with respect to the regions of gaps opened previously. Since those gaps fall to the right of the current u_n because of the greedy selection of previous u_i, their active regions contain r_{u_n}.

Gaps $g_{u_1}, \ldots, g_{u_n}$ are introduced in the place of u_1, …, u_n, and the length of $s \in \text{opt}$ that was unused in the shifting scheme, if positive, is divided into contiguous fragments of lengths $l_{u_1}, \ldots, l_{u_n}$. We note that the division must use all of the unused length. The fragments are identified with sensors $u_1, \ldots, u_n \in \text{app}_C$ respectively. The old u_i in app_C and the "new" u_i in opt_C are relabeled uniquely as $v_1, \ldots v_n$ to avoid later confusion with the gaps $g_{u_1}, \ldots g_{u_n}$ and their active regions.

Each new sensor v_i is assigned weight $w_{v_i} = w_{u_i}$. Since each u_i followed s in opt, this can be assumed over all u_i without increasing the cost of opt_C, by Lemma 5.2. Clearly, $p_{v_i,\text{opt}_C} \leq p_{s,\text{opt}_C}$ and $p_{v_i,\text{app}_C} = p_{v_i,\text{app}_C}$, so any charges made against the v_i in opt_C under the usual constraints cannot exceed the original cost of s in opt.

Invariant 5.11 is maintained, under the modified subroutine of Proposition 5.10. It is clear that Invariant 5.1 also holds.

Case 2: $s \notin \text{app}_C \cap \text{opt}_C$.

If the priority queue is not empty, we use Invariant 5.11 to draw viable sensors from the priority queue to pay for s.

Suppose we can entirely pay for s using auxiliary sensors drawn from the queue. Since $p_{t,\text{opt}_C} \leq x$ for all $t \in PQ$, and t is ordered maximally by p_{t,opt_C}, we have that the sensors t' remaining in the queue after the cost of s is paid satisfy $p_{t',\text{opt}_C} \leq x - l_s$, since every positive p_{t,opt_c} is unique.

If the auxiliary sensors in the priority queue can only pay for a partial length of s (or no length of s, meaning the queue is empty), we fall back on the argument given in case $n = 1$, which does not rely on the assumption that s is the only sensor left in app_C.

In either case, Invariant 5.11 is preserved and the gaps tagged by the secondary sensors drawn from the queue are compacted against by the length of those sensors. The remaining regions and gaps are unaffected except as a result of the compaction that occurred after drawing from the queue. Due to the ordering of the queue, the only effect this has is to reduce or close the rightmost gaps, leaving Invariant 5.12 intact. □

Theorem 5.14 (Algorithm 10 is a 2-approximation to WeightedMinSum)**.** *Let app be a solution to an instance of the WeightedMinSum problem with cost APP and let opt be any* *fixed optimal solution for the same instance with cost OPT. Then $APP \leq 2OPT$.*

Proof. We invoke Proposition 5.13, noting that all three Invariants are trivially satisfied by the solution computed by Algorithm 10. □

Finally, we show that no smaller approximation bound holds for Algorithm 10.

Theorem 5.15 (Tightness of the approximation bound)**.** *The 2-approximation bound for Algorithm 10 is tight.*

Proof. We construct a parametric family of covering problems whose solution costs under Algorithm 10 are arbitrarily close to twice their optimal solution costs.

Let $L > 0$ be the length of the barrier. Lying disjointly to the left of the barrier are two sensors. The first sensor, s_1, has length $l_1 = cL$ and distance from the line $d_1 = 1 - \varepsilon$ for some $\varepsilon > 0$ and $1 - \varepsilon < c < 1$, with weight $w_1 = c$.

Similarly, sensor s_2 has $l_2 = L$, $d_2 = c$ and $w_2 = 1$.

It is readily seen that the optimal solution consists only of s_2, so that $OPT = c + L$.

Running Algorithm 10 on the instance produces an ordered solution of $< s_2, s_1 >$, with cost $c(1-\varepsilon) + cL + c + L - cL = c(1-\varepsilon) + c + L$, giving the approximation ratio

$$\frac{c(1-\varepsilon)+c+L}{c+L} = \frac{1-\varepsilon}{1+L/c} + 1$$

Let $L = c\varepsilon$. Since we have free choice of $\varepsilon > 0$ and $\lim_{\varepsilon \to 0^+}(1-\varepsilon)/(1+\varepsilon) = 1$, the performance ratio can come as close to 2 as we wish, demonstrating that the bound is tight. □

5.3 An FPTAS for the one-sided disjoint weighted case

Initially we develop an FPTAS for the left lying disjoint weighted barrier coverage problem, then extend it to a more general two-sided version.

The idea is that the one-sided FPTAS considers sensors for placement one at a time, in the order specified by Lemma 5.2, the Order Preservation Property. Packing sensors leftward from the barrier point L, it maximizes the length of a partial barrier coverage without exceeding a given budget in sensor movements.

Placements are structured recursively through dynamic programming. We define a dynamic programming table $f^*(i,z)$, where i restricts the placement to the first i sensors in the ordering of Lemma 5.2, and z sets the budget to be spent in the movement of sensors.

As a recurrence relation, $f^*(i,z)$ is defined as

$$f^*(i,z) = \max\{\min\{f^*(i-1,z), g^*(i,z)\}, 0\}$$

where for $i > 0$ and $z \geq 0$,

$$g^*(i,z) = \min_{0 \leq x \leq z} \{f^*(i-1,z-x) - 2r_i : w_i \cdot |f^*(i-1,z-x) - r_i - x_i| \leq x\}$$

Intuitively, in the top-level recurrence, we either include the sensor s_i in the partial coverage or we do not, out of preference for some cheaper placement involving only the first $i-1$ sensors. If we do place s_i, we assign some budget $0 \leq x \leq z$ to pay for the movement of s_i, and use the remaining budget $z-x$ to pay for other $i-1$ sensors. This proceeds under the assumption that the total cost of moving s_i, $|f^*(i,z-x) - r_i - x_i|$, does not exceed x. Then the value of $f(i,z)$ is set to $f^*(i-1),z-x) - 2r_i$, the left end point of the partial cover, provided that the inclusion of s_i enables us to cover more of the barrier.

From the definition of $f^*(i,z)$, we have

$$OPT = \min_{z \geq 0}\{z : f^*(n,z) \leq 0\}$$

where n is the number of sensors in S. From here, it is a routine exercise in induction to show that the last equation is correct.

To translate $f^*(i,z)$ into a recurrence we can compute, two things must happen. First, we note that the budget z is assumed to be any non-negative real number. Since most real numbers cannot be represented precisely on a binary computer, z must be restricted to a discrete sampling of the real numbers. Second, for the computation to terminate, we also need to induce an upper bound on the budgets z.

The upper bound on budgets Z will be defined as the cost of the solution offered by Algorithm 10, and immediately we get $Z \leq 2 \cdot OPT$ from the work of the preceding section. The discrete sampling of budgets z will be given by iterating across the interval $[0, Z]$ in integer multiples of the quantum $\zeta = \varepsilon Z / 2(n+1)$, which depends on some $\varepsilon > 0$.

We modify the recurrence to reflect the discretization as follows.

$$f(i, k\zeta) = \min\{f(i-1, k\zeta), f(i-1, (k-1)\zeta), g(i, k\zeta)\}$$

where

$$\begin{aligned} g(i, k\zeta) \quad = \quad & \min_{1 \leq c \leq k} \{f(i-1, (k-c)\zeta) - 2r_i \\ & \text{s.t. } w_i \cdot |f(i-1, (k-c)\zeta) - r_i - x_i| \leq c\zeta\} \end{aligned}$$

We consider $f(i, k\zeta) = L$ if $i = 0$ or $k = 0$. The claim that an FPTAS can be defined on the basis of f is founded on the following lemma.

Lemma 5.16. *For every $1 \leq i \leq n$, and every integer $k \geq 0$, there is a barrier coverage of the interval $(f(i, k\zeta), L]$ that uses the first i sensors and has movement cost no more than $k\zeta$.*

Conversely, for any $x \geq 0$, suppose there is a barrier coverage of the interval $[x, L]$ that uses the first i sensors with total movement cost no more than z. Let k be an integer satisfying $k \cdot \zeta \geq z$. Then $f(i, (k+i)\zeta) \leq x$.

Proof. Both claims of the lemma can be proved inductively, and we begin by proving the first claim.

For $i = 1$, if $k\zeta \geq w_1(d_1 + L)$, then by the definition of f, $f(1, k\zeta)$ is the left end point of the cover consisting only of s_1 right-aligned to L, attained using movement cost no more than $k\zeta$. Otherwise, no movement of s_1 is possible, and $f(1,k\zeta) = L$; then $(f(1,k\zeta),L] = \emptyset$ and the statement is vacuously true.

Now let $i = m+1$ and suppose the statement holds in the case $i = m$. In the case that i is not selected for use in the cover, we have $f(i,k\zeta) = f(i-1,k\zeta)$ and apply the induction hypothesis to the first $i-1$ sensors. Otherwise, $f(i,k\zeta) = f(i-1,(k-c)\zeta) - 2r_i$ for some $1 \leq c \leq k$, where the cost of moving s_i into place is no more than $c\zeta$.

By the induction hypothesis, there is a cover involving the first $i-1$ sensors of the barrier interval $(f(i-1,(k-c)\zeta),L]$ of cost $(k-c)\zeta$. It follows from the definition of the case selected in f that this covering can be extended by length $l_i = 2r_i$ using sensor s_i, at a cost of at most $c\zeta$, for a total cost of $k\zeta$.

For the second claim, we again use induction. Let $x \geq 0$ and $i = 1$ such that there is a barrier coverage of $[x,L]$ using only the first i sensors. Since we are limited to the exclusive use of the first sensor, we must have $x \geq L - l_1$. Let $z = w_1(d_1 + L)$, the cost of moving s_1 to right-align with L. Let $k \in \mathcal{Z}$ such that $k\zeta \geq z$. Then $f(1,(k+1)\zeta) \leq x$, since $f(1,(k+1)\zeta) = L - l_1$.

Now let $i = m+1$ for some $m \geq 1$ and suppose the induction hypothesis holds for every $x \geq 0$ satisfying the stated condition when $i = m$. Let $x \geq 0$ and suppose there is a cover of the interval $[x,L]$ from among the first $(i+1)$ sensors. If the cover is in fact limited to the first i sensors, we apply the induction hypothesis, noting that due to budget increases, $f(i,(k+i+1)\zeta) \leq f(i,(k+i)\zeta) \leq x$. Combined with $f(i+1,(k+i+1)\zeta) \leq f(i,(k+i+1)\zeta)$, we get the desired statement.

Otherwise, s_{i+1} is included in the cover. Due to Lemma 5.2, we can freely suppose that s_{i+1} lies closest to 0 of all sensors in the cover at no additional cost. Then lying to the right of s_{i+1}, there is a cover of the interval $(x + l_{i+1}, L]$ using the first i sensors, of cost z. By the induction hypothesis, it follows that for every $k \in \mathcal{Z}$ satisfying $k\zeta \geq z$, $f(i,(k+i)\zeta) \leq x + l_{i+1}$.

Select an integer k' satisfying $k'\zeta \geq w_{i+1}(d_{i+1}+x+l_{i+1})+z$, the total cost of the cover with s_{i+1}, and an integer $k'' \leq k'$ satisfying $(k''-1)\zeta < w_{i+1}(d_{i+1}+x+l_{i+1}) \leq k''\zeta$. Then $(k'-k''+1)\zeta \geq z$, the cost of moving the first i sensors into place. Setting $c = k''$, we see that $f(i,(k'-k''+i+1)\zeta)-2r_{i+1}$ belongs to the range of values of which $g(i+1,(k'+i+1)\zeta)$ is the minimum. We observe also that by the induction hypothesis, $f(i,(k'-k''+i+1)\zeta) \leq x+l_{i+1}$, and therefore, $f(i,(k'-k''+i+1)\zeta)-2r_{i+1} \leq x$. $f(i+1,k'\zeta+i+1) \leq g(i+1,k'\zeta+i+1)$ by definition of f, and this completes the proof of the induction step. □

The pseudocode for the one-sided FPTAS is given as Algorithm 11, which iteratively calculates the recurrence of f, storing intermediary results in a dynamic programming table.

Algorithm 11 The FPTAS for one-sided disjoint MinSum

```
1: procedure LeftFPTAS($x, r, L, \varepsilon$)
2:     $Z \leftarrow$ a value in $[OPT, n \cdot OPT]$ computed as described above
3:     $\zeta = \varepsilon Z/2(n+1)$
4:     $n \leftarrow$ # of sensors
5:     for $k$ from 0 to $\ldots Z/\zeta + n + 1$ do
6:         $f[0, k\zeta] \leftarrow L$
7:     for $i$ from 1 to $n$ do
8:         for $k$ from 0 to $Z/\zeta + n + 1$ do
9:             $g[i, k\zeta] \leftarrow +\infty$

10:            for $c$ from 0 to $k$ do
11:                if $|f[i-1, k\zeta - c\zeta] - r_i - x_i| \leq c\zeta$ then
12:                    endpt $\leftarrow f[i-1, k\zeta - c\zeta] - 2r_i$
13:                    $g[i, k\zeta] \leftarrow \min(g[i-1, k\zeta], \text{endpt})$

14:            $f[i, k\zeta] \leftarrow \min(f[i-1, k\zeta], g[i, k\zeta])$
15:            if $k > 0$ then
16:                $f[i, k\zeta] \leftarrow \min(f[i, k], f[i, (k-1)\zeta])$
17:     return $\min\{k\zeta : k \leq Z/\zeta + n + 1, f(n, k\zeta) \leq 0\}$
```

Recall that Z is the cost of the solution produced by Algorithm 10, so that $Z \leq 2 \cdot OPT$. We prove that Algorithm 11 is an FPTAS.

Theorem 5.17. *Given* $\varepsilon > 0$, *Algorithm 11 computes a* $(1+\varepsilon)$*-approximation in* $O(n^3/\varepsilon^2)$.

Proof. Algorithm 11 determines k^*, which is defined as

$$k^* = \min\{k\zeta : 0 \leq k \leq \frac{Z}{\zeta} + n + 1 \text{ and } f(n, k\zeta) \leq 0\}$$

Let k' be the smallest $k' \in \mathcal{Z}$ such that $(k'-1)\zeta < OPT \leq k'\zeta$. Since $Z + (n+1)\zeta \geq OPT + (n+1)\zeta \geq (k'+n)\zeta$, $k'+n$ is in the range of integers $0 \leq k \leq Z/\zeta + (n+1)$. We have by Lemma 5.16 that there is a full coverage of $[0, L]$ using the n sensors given, and that its cost is no more than $(k'+n)\zeta$.

From these observations and the definition of k^*, we have immediately that $k^* \leq k' + n$. Furthermore, we have

$$k^*\zeta \leq (k'+n)\zeta \leq OPT + (n+1)\zeta \leq OPT + \frac{\varepsilon Z}{2} \leq (1+\varepsilon) \cdot OPT$$

Lastly, by multiplying the containing loop ranges together, Algorithm 11 has runtime $O(n \cdot (Z/\zeta + n + 1)^2) = O(n^3/\varepsilon^2)$. □

5.4 An FPTAS for the two-sided disjoint MinSum problem

We will adapt Algorithm 11 to give a generalized FPTAS for the two sided, disjoint MinSum problem. First, we will prove some structural properties for later use in the two-sided FPTAS.

Lemma 5.18. *For any optimal complete barrier coverage of a two-sided disjoint MinSum instance, we may suppose without loss of generality that there exists a point $x^* \in [0, L]$ such that the interval $[0, x^*)$ is covered only by sensors s_i with $x_i < 0$ and $(x^*, L]$ is covered only by sensors s_i with $x_i > L$.*

Proof. Fix any optimal solution of a given two-sided disjoint MinSum instance and suppose that the statement of the lemma does not hold. That is, there exist sensors s_l with $x_l < 0$ and s_r with $x_r > 0$. Since s_r initially lies to the right of the barrier, its movement cost m_r is negative, and will be added to x_r to obtain its position in the coverage. So that the statement of the lemma fails, we suppose $x_l + m_l > x_r + m_r$.

We may suppose that s_l and s_r are consecutive, since this must be true of some pair of s_l and s_r for the property to fail. We swap s_l and s_r. Since s_l crossed the point $x_r + m_r$ on its way to $x_l + m_l$, the swap has reduced m_l, and likewise for m_r. Since the original solution had optimal cost and the swapping actions could not have increased it, the cost remains optimal. □

Another property we will exploit is the fact that overhang can be eliminated on one side of an optimal solution. A sensor s_i overhangs from the right side of a barrier if $x_i + m_i - r_i < L < x_i + m_i$ and similarly if it overhangs from the left side.

Lemma 5.19. *There exists an optimum barrier coverage in which there is either no left overhang or no right overhang.*

Proof. Fix a solution of optimal cost and assume that both left and right overhang are present. Define the sensor sets $S_L = \{s_i \in \text{opt} : x_i < 0\}$ and $S_R = \{s_i \in \text{opt} : x_i > L\}$. Suppose that $\sum_{s \in S_L} w_s \geq \sum_{t \in S_R} w_t$. Then shifting the packed sensors of opt simultaneously to the left by the length of the right overhang changes the changes of the solution, adding to it the non-positive amount

$$\sum_{t \in S_R} w_t - \sum_{s \in S_L} w_s \leq 0$$

If $\sum_{s \in S_L} w_s < \sum_{t \in S_R}$, the same argument holds by symmetry. □

To get a two-sided disjoint MinSum FPTAS, it will suffice to limit ourselves to solutions without either left- or right overhang, where the left and right sensors are partitioned to either side of a unique point $0 \leq x^* \leq L$.

Theorem 5.20. *There is an FPTAS for the two-sided, disjoint MinSum problem with running time $O(\frac{n^5}{\varepsilon^3})$.*

Proof. The FPTAS proceeds in two phases. First, we pack from the right using only sensors that lie initially to the right of L, growing positively in the direction towards 0. This will require us to reverse the order of selection in the one-sided case, in preference of the sensors s of *smallest* $ratio(s) = w_s/l_s$.

We'll designate the cost of such a packing using the first i right-lying sensors of budget z as $f_R(i,z)$. Its definition is given as the recurrence relation

$$f_R(i,z) = \max\{f_R^*(i-1,z), g_R^*(i,z)\},$$

where

$$g_R^*(i,z) = \max_{x\in[0,z]}\{f_R^*(i-1,z-x) - 2r_i : w_i\cdot(x_i - f_R^*(i-1,z-x) + r_i) \leq x\}.$$

$f_R(i,z)$ is structured as a mirror version of the one-sided recurrence $f^*(i,z)$. Once again, the strategy we take is to discretize f_R. In its discretized form, $f_R(i,z)$ will serve as an estimate of the left-right partition point x^*, whose existence can be assumed in an optimal solution, as shown in Lemma 5.18. With $f_R(i,z)$ serving as the right endpoint of a barrier, we will run the original one-sided disjoint MinSum FPTAS on the interval $[0, f_R(i,z)]$. This will occur for every combination of $0 \leq i \leq n$ and integer multiple of some $\zeta \leq Z$, where both ζ and Z have yet to be chosen. The result will be a solution of the type described in Lemma 5.18, without any right overhang. Then we will run the same procedure with swapped sides, to produce solutions without any left overhang, and take the minimum cost solution over both. This will yield an FPTAS with approximation ratio $1+\varepsilon$ and run time $O(n^5/\varepsilon^3)$.

We begin by computing Z, as follows. For each sensor s_i, let $\alpha_i = \min(|x_i|, |x_i - L|)$. We try to guess the value of $\alpha = \max_{s_i\in\text{opt}}\alpha_i$ by iterating through the n sensors given. For each guess α_i, move all sensors s_j with $\alpha_j \leq \alpha_i$ to align with 0 or L, whichever is closest, at their nearest endpoint. For the correct guess α_i, the total movement so far is no more than

n times the total movement used in the optimal solution to move sensors from their starting points to the endpoints of $[0,L]$. This is because s_i is included in opt supposing that α_i is the correct guess, and we choose an additional $i-1 \leq n-1$ sensors whose distance to their nearest endpoint is no more than s_i's.

Now guess the number i_L of left-lying sensors used in the optimal solution. Of the sensors moved to 0 in the previous step, choose the top i_L of them according to least $ratio(\cdot)$. Where the selected sensors have total length $\mathcal{L}$, greedily cover $[0,\mathcal{L}]$ with them, and greedily cover $[\mathcal{L},L]$ using the right-lying sensors aligned at L, again chosen by least $ratio(\cdot)$. Due to the greedy selection by least $ratio(\cdot)$, the movement cost of this second step is no more than the total movement of sensors that occurs within the optimal solution when moving within the barrier interval $[0,L]$.

If we total all movements made in this solution into Z, we have $OPT \leq Z \leq n \cdot OPT$. We set $\zeta = \varepsilon Z/n(n+1)$ for a given $\varepsilon > 0$. We will now discretize f_R^* over integer multiples of ζ, as we did in the one-sided disjoint MinSum FPTAS, into

$$f_R(i,k\zeta) = \max\{f_R(i-1,k\zeta), g_R(i,k\zeta)\}$$

where

$$g_R(i,k\zeta) = \max_{1\leq c\leq k}\{f_R(i-1,(k-c)\zeta)+2r_i : w_i \cdot (x_i - f_R(i-1,(k-c)\zeta)+r_i) \leq c\zeta\}$$

and

$$f_R(0,k\zeta) = 0 \text{ for all } k$$

where, as before, we iterate over k.

We introduce a claim similar to the one made in Lemma 5.16.

Lemma 5.21. *For every $0 \leq i \leq n$ and every integer $0 \leq k$, there is a barrier for the interval $[f_R(i,k\zeta),L]$ that uses the top i sensors to the right of L, chosen by greatest ratio$(\cdot)$, and* *has movement cost at most $k \cdot \zeta$.*

Conversely, for $x \geq 0$ suppose there is a barrier for an interval $[x, L]$ that uses the top i sensors, chosen by greatest ratio$(\cdot)$, to the right of L with total movement cost at most z. Let $k \in \mathcal{Z}$ be such that $k \cdot \zeta \geq z$. Then $f_R(i, (k+i)\zeta) \leq x$.

The proof is identical to that of Lemma 5.16.

Let x^* be the midpoint in the optimal solution, so that $[0, x^*]$ is covered entirely by sensors s_i with $x_i < 0$, and $[x^*, L]$ covered entirely by sensors s_i with $x_i > L$. Let OPT_L and OPT_R denote the costs of the solutions that make up either side of the optimal solution.

Let k_R^* be the smallest integer satisfying $k_R^*\zeta \geq OPT_R + (n_R + 1)\zeta$. By Lemma 5.21, $f_R(n_r, k_R^*)\zeta) \leq x^*$ and there is a covering of $[f_R(n_r, k_R^*\zeta), L]$ with cost no greater than $k_R^*\zeta$.

We approximate the left hand side solution with cost OPT_L by packing the interval $[0, f_R(n_R, k_R^*\zeta)]$ with left-lying sensors, using the FPTAS of the previous section. By the previous section's results, the interval $[0, x^*]$ can be covered at cost OPT_L. Additionally, $f_R(n_R, k_R^*\zeta) \leq x^*$. Thus, the total cost of the solution covering $[0, L]$ by splitting it into two sub-intervals around some point $f_R(n_R, k'\zeta)$ is at most

$$\begin{aligned} & (1+\varepsilon)OPT_L + OPT_R + (n_R+1)\zeta \\ \leq \;& (1+\varepsilon) \cdot OPT + (n+1)\zeta \\ \leq \;& (1+2\varepsilon) \cdot OPT \end{aligned}$$

Since we do not know the explicit value of x^*, we must guess. We guess for x^* by iterating over the $f(n_R, k_R\zeta)$ in the pseudocode of the two-sided FPTAS, whose pseudocode is given here. The process of finding all values of $f_R(n_R, k_R\zeta)$ is structured after the pseudocode of Algorithm 11, but reflects the discretized recurrence of f_R^* rather than that of f_L^*.

The two-sided FPTAS has run time $O(n^5/\varepsilon^3)$. Finding all the f_R values takes time $O(n^5/\varepsilon^3)$, and each call to Algorithm 11 takes time $O(n^3/\varepsilon^2)$. There are $O(n^2/\varepsilon)$ calls to

Algorithm 12 The FPTAS for *DisjointMinSum*

```
1: procedure DisjointMinFPTAS($(x, r, L)$)
2:     $Z \leftarrow$ a value in $[OPT, n \cdot OPT]$ computed by the $n$-approximation algorithm
3:     $\zeta \leftarrow \varepsilon Z/(n(n+1))$
4:     $n_R \leftarrow$ # of right-sensors
5:     $x^L, r^L \leftarrow$ the sub-lists of positions and ranges $x, r$ for the left-sensors

6:     for $k$ from 0 to $\ldots Z/\zeta + n + 1$ do
7:         $f_R[0, k\zeta] \leftarrow L$
8:     for $i$ from 1 to $n$ do
9:         for $k$ from 0 to $Z/\zeta + n + 1$ do
10:            $g_R[i, k\zeta] \leftarrow +\infty$

11:            for $c$ from 0 to $k$ do
12:                if $w_i \cdot |x_i - f_R[i-1, k\zeta - c\zeta] + r_i| \leq c\zeta$ then
13:                    endpt $\leftarrow f_R[i-1, k\zeta - c\zeta] + 2r_i$
14:                    $g_R[i, k\zeta] \leftarrow \min(g_R[i-1, k\zeta], \text{endpt})$

15:            $f_R[i, k\zeta] \leftarrow \min(f_R[i-1, k\zeta], g_R[i, k\zeta])$
16:            if $k > 0$ then
17:                $f_R[i, k\zeta] \leftarrow \min(f_R[i, k], f_R[i, (k-1)\zeta])$
18:    best $\leftarrow +\infty$
19:    for $k_R$ from 0 to $Z/\zeta + n + 1$ do
20:        left $\leftarrow$ LeftFPTAS$(x^L, r^L, f_R(n_R, k_R\zeta), \varepsilon)$
21:        best $\leftarrow \min(\text{best}, \text{left} + k_R\zeta)$
22:    return best
```

Algorithm 11 in the final loop, giving a total run time of $O(n^5/\varepsilon^3)$. $\square$

Chapter 6

Conclusion

In this dissertation, we established approximation bounds for two classes of covering problems, and provided optimal algorithms for some special cases.

First, approximation algorithms were provided for the uniform, maximum and set cover CVC problems. Additionally, we described an exact algorithm for maximum uniform CVC on the line, which runs in polynomial time under the assumption that the number of range/-capacity pairs is fixed.

Second, we have shown that the barrier coverage FPTAS of [6] can be extended to the weighted case of the disjoint MinSum barrier coverage problem. By developing a 2-approximation algorithm for the weighted disjoint MinSum, the time and space complexities of our FPTAS decrease from those of the unweighted FPTAS by quadratic and linear factors respectively.

6.1 Future Work

Other variants of the barrier coverage problem remain to be solved. The MinMax barrier coverage problem on the line seeks to cover a barrier with sensors, as in the MinSum problem. The only change is to the objective function, which now assumes the greatest cost of any individual sensor movement. Can mitigating constraints be imposed on MinMax to allow a constant factor approximation algorithm?

Bibliography

[1] Selim Akl, Robert Benkoczi, Daya Ram Gaur, Hossam Hassanein, Shahadat Hossain, and Mark Thom. On a class of covering problems with variable capacities in wireless networks. *Theoretical Computer Science*, 575:42–55, 2015.

[2] Aaron M. Andrews and Haitao Wang. Minimizing the aggregate movements for interval coverage. In *Algorithms and Data Structures: 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 28–39, Cham, 2015. Springer International Publishing.

[3] Paul Balister, Béla Bollobas, Amites Sarkar, and Santosh Kumar. Reliable density estimates for coverage and connectivity in thin strips of finite length. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, MobiCom '07, pages 75–86, New York, NY, USA, 2007. ACM.

[4] Amotz Bar-Noy, Dror Rawitz, and Peter Terlecky. Maximizing barrier coverage lifetime with mobile sensors. In *Algorithms–ESA 2013*, pages 97–108. Springer, 2013.

[5] Amotz Bar-Noy, Dror Rawitz, and Peter Terlecky. "green" barrier coverage with mobile sensors. In *Algorithms and Complexity: 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*, pages 33–46, Cham, 2015. Springer International Publishing.

[6] Robert Benkoczi, Zachary Friggstad, Daya Gaur, and Mark Thom. Minimizing total sensor movement for barrier coverage by non-uniform sensors on a line. In *Algorithms for Sensor Systems*, pages 98–111. Springer, 2015.

[7] Oded Berman, Zvi Drezner, and Dmitry Krass. Generalized coverage: New developments in covering location models. *Computers and Operations Research*, 37(10):1675 – 1687, 2010.

[8] Daniel Catrein, Lorenz A. Imhof, and Rudolf Mathar. Power control, capacity, and duality of uplink and downlink in cellular CDMA systems. *IEEE Transactions on Communications*, 52(10):1777–1785, 2004.

[9] Ai Chen, Santosh Kumar, and Ten H. Lai. Designing localized algorithms for barrier coverage. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, MobiCom '07, pages 63–74, New York, NY, USA, 2007. ACM.

[10] Danny Z Chen, Yan Gu, Jian Li, and Haitao Wang. Algorithms on minimizing the maximum sensor movement for barrier coverage of a linear domain. *Discrete & Computational Geometry*, 50(2):374–408, 2013.

[11] Richard Church and Charles ReVelle. The maximal covering location problem. *Papers of the Regional Science Association*, 32(1):101–118, 1974.

[12] Vasek Chvatal. *Linear programming*. Macmillan, 1983.

[13] J. Czyzowicz, E. Kranakis, D. Krizanc, I. Lambadaris, L. Narayanan, L. Stacho, J. Urrutia, and M. Yazdani. On minimizing the maximum sensor movement for barrier coverage of a line segment. In *Proceedings of 8th International Conference on Ad Hoc Networks and Wireless*, pages 22–25, 2002.

[14] Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, Ioannis Lambadaris, Lata Narayanan, Jaroslav Opatrny, Ladislav Stacho, Jorge Urrutia, and Mohammadreza Yazdani. On minimizing the sum of sensor movements for barrier coverage of a line segment. In *Ad-Hoc, Mobile and Wireless Networks*, pages 29–42. Springer, 2010.

[15] Stefan Dobrev, Stephane Durocher, Mohsen Eftekhari, Konstantinos Georgiou, Evangelos Kranakis, Danny Krizanc, Lata Narayanan, Jaroslav Opatrny, Sunil Shende, and Jorge Urrutia. Complexity of barrier coverage with relocatable sensors in the plane. *Theoretical Computer Science*, 579:64 – 73, 2015.

[16] Zvi Drezner and Horst W Hamacher. *Facility location: applications and theory*. Springer Science & Business Media, 2001.

[17] Michael R Fellows, Jan Kratochvíl, Martin Middendorf, and Frank Pfeiffer. The complexity of induced minors and related problems. *Algorithmica*, 13(3):266–282, 1995.

[18] Stephen Hanly. Congestion measures in DS-CDMA networks. *IEEE Transactions on Communications*, 47(3):426–437, 1999.

[19] Harri Holma and Antti Toskala. *WCDMA for UMTS: HSPA Evolution and LTE*. Wiley, fourth edition, 2007. ISBN: 978-0470319338.

[20] Santosh Kumar, Ten H Lai, and Anish Arora. Barrier coverage with wireless sensors. In *Proceedings of the 11th annual international conference on Mobile computing and networking*, pages 284–298. ACM, 2005.

[21] Pitu B Mirchandani and Richard L Francis. *Discrete location theory*. 1990.

[22] Prakash Mirchandani, Rajeev Kohli, and Arie Tamir. Capacitated location problems on a line. *Transportation Science*, 30(1):75–80, 1996.

[23] Hasan Pirkul and David A. Schilling. The maximal covering location problem with capacities on total workload. *Management Science*, 37(2):pp. 233–248, 1991.

[24] Ayman Radwan and Hossam Hassanein. Capacity enhancement in CDMA cellular networks using multi-hop communication. In *Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 832–837. IEEE, 26-29 June 2006.

[25] Y. Hung Tam, Hossam S. Hassanein, Selim G. Akl, and Robert Benkoczi. Optimal multi-hop cellular architecture for wireless communications. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 738–745, Nov. 2006.

[26] Yang Wang, Shuang Wu, Xiaofeng Gao, Fan Wu, and Guihai Chen. Minimizing mobile sensor movements to form a line k-coverage. *Peer-to-Peer Networking and Applications*, pages 1–16, 2016.

[27] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.