EXACT AND APPROXIMATE ALGORITHMS FOR SOME COMBINATORIAL PROBLEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Soroush Hosseini Alamdari

May 2018

EXACT AND APPROXIMATE ALGORITHMS

FOR SOME COMBINATORIAL PROBLEMS

Soroush Hosseini Alamdari, Ph.D.

Cornell University 2018

Three combinatorial problems are studied and efficient algorithms are presented for each of them. The first problem is concerned with lot-sizing, the second one arises in exam-scheduling, and the third lies on the intersection of the k-median and k-center clustering problems.

BIOGRAPHICAL SKETCH

Soroush Hosseini Alamdari was born in Malayer, a town in Hamedan province of Iran right on the outskirts of Zagros mountains. After receiving a gold medal in the Iranian National Olympiad in Informatics he went to Sharif University of Technology in Tehran to study Computer Engineering. After completing his BSc he went to the University of Waterloo and received his MMath in computer science. Then after a short period of working in industry, he came back to complete his PhD in computer science here at the Cornell University.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my Adviser Prof. David Shmoys who accepted me as a student, taught me so many fascinating things, and supported me in whatever endeavor I took on. Without his care and attention this document would simply not be possible.

I would also like to thank all my friends, colleagues and co-authors. Regarding this thesis I am particularly thankful to Chaoxu Tong with whom I closely worked on some facility location problems that lead to an unpublished paper with Chaitanya Swamy and David Shmoys, some of the results of which are presented in Chapter 3 of this document.

I am also thankful to Carla Gomes and Jon Kleinberg for accepting to be on my thesis committee, and to Bart Selman for following on my progress during my PhD studies here at Cornell University.

Lastly, I would like to thank my wonderful wife Christina who never ceased to trust in me. Most humbling, she bore me two precious daughters, Mana and Aaliyah, for which I find myself unable to ever thank her justly.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRO

In this chapter we briefly review each of the three problems and outline our contributions in each case.

1.1 Lot-sizing with hierarchical types

Suppose we are running an operation over a time horizon when certain processes are scheduled, each demanding a set of resources. We are able to procure resources at any time with varying prices and store them with a cost that is increasing relative to the duration of storage. To satisfy the demand of a certain process we would need to have the appropriate resources available at the time that process is scheduled for. This problem is known as Lot-sizing.

Suppose that our resources have types and that some types can be used in place of others without extra penalty. Particularly, we consider cases where the structure of the replacement relationship between the types constitutes a rooted tree (See Chapter 2 for definition,) in the sense that each resource type can be used to satisfy a demand for a any of its descendants type.

For the problem of planning the acquisition of necessary resources over the time horizon we present an efficient algorithm that finds an optimal schedule, minimizing the total cost of purchases and storage. We also analyze some of the theoretical aspects of the problem regarding its natural linear relaxation (See Chapter 2 for definition.)

1.2 Exam scheduling

Suppose we are scheduling the final exams for a large school, given the data regarding the exams each students must take. For example, in the case of Cornell University we have about 20 thousand students taking about 700 different exams at the end of each semester in a period of roughly one week. In each day there are a number of disjoint time-slots where exams are taken. For example, in the case of Cornell University, there are about 20 time-slots in total during the exam period where exams can be scheduled, and the goal is to ensure that no students end up with two overlapping exams, or too many back-to-back exams consecutively.

For this problem we present mixed-integer formulations (See Chapter 2 for definition) that are able to produce conflict-free schedules within reasonable time constraints. More specifically, we present a two phase algorithm that first colors the exams and then assigns each color to a time-slot in order to minimize undesirable cases such as an student with 3 exams in a consecutive 24 hours.

1.3 Bi-criteria approximation for k-center and k-median

k-center and k-median are clustering techniques that are widely used and extensively studied. Both problems require us to come up with k clusters to represent a given set of n points, but while k-center is concerned with the largest radius among the clusters, k-median is concerned with the total distance of the points from the center of their clusters.

We study the intersection of the two problems and present an algorithm

that balances between the two objectives, producing solutions that are approximately optimal relative to a combination of the two objectives.

We generalize our approach to the online case where one has to select cluster centers incrementally and k is only revealed after k centers are chosen.

CHAPTER 2

INTRODUCTION TO FUNDAMENTALS

Combinatorics is a sub-field of mathematics concerned with objects and relations between them. For example, sorting a set of objects based on a given parameter is a combinatorial task. In this thesis we will consider some combinatorial optimization problems and provide approximate and exact algorithms for them that run in polynomial time.

Polynomial algorithm: An algorithm runs in polynomial time if the number of iterations it takes to produce its output is bounded by a polynomial relative to the size of the input. For example, if our sorting algorithm finds the solution for a set of n objects by considering all orderings, then we have a non-polynomial sorting algorithm since the number of orderings of n objects is $n!$ which cannot be bounded by any polynomial in terms of n.

The O notation: To give a more precise bound on the run-time of an algorithm we use the O notation. When the number of iterations that an algorithm takes can be bounded by $cf(n)$ for some constant c, we say that the algorithm runs in $O(f(n))$. For example, the sorting algorithm that considers all orderings and chooses one that is sorted would have a running time that can be bounded by $O(n \times n!)$, because there are $n!$ orderings and one can check if each of them is sorted in cn iterations for some constant c.

Optimization problem: An optimization problem presents us with a set of choices that come together as a decision profile and for each decision profile there is a cost, and the objective is to make these choices in polynomial time such that the cost is minimized. For example, we can formulate our sorting

example as optimization problem where the decision profile corresponds to an ordering, and the cost of an ordering is 0 if it is sorted and 1 otherwise. Then minimizing the cost would be equivalent to finding an ordering of the given objects that is sorted.

NP-Hardness: In some cases it is impossible to find an algorithm that is able to find an optimum solution in polynomial time. One notable such a class of problems are the NP-complete problems that are widely believed to be impossible to solve efficiently, although there is no proof for this impossibility. NP-hard problems are those problems that are at least as hard as the NP-complete ones, and therefore for them there is no polynomial time solution as well.

Approximation algorithm: For NP-hard problems there is practically no hope of devising polynomial algorithms that would produce optimum solutions, but one can hope for devising algorithms that are approximately optimal. More precisely, if an algorithm is guaranteed to produce solutions of cost $\alpha\mathrm{OPT}$ where OPT is the optimal cost, we refer to that algorithm as an α approximation for the problem.

Bicriteria approximation: Consider a problem with two separate objective functions f_1 and f_2, and suppose that OPT is an ideal solution with costs $f_1(\mathrm{OPT})$ and $f_2(\mathrm{OPT})$. Suppose we have an algorithm that given $f_1(\mathrm{OPT})$ is able to produce a solution X with objectives $f_1(X)$ and $f_2(X)$ such that $f_1(X) \leq \alpha f_1(\mathrm{OPT})$ and $f_2(X) \leq \beta f_2(\mathrm{OPT})$. Then this algorithm is a (α, β)-bicriteria approximation for f_1 and f_2.

2.1 Graph Theory

A graph is a combinatorial object comprised of a set of vertices and a set of edges, where each edge has two vertices as its ends and its function is to connect them. We use $G(V, E)$ to represent a graph G with vertex set V and edge set E. An edge e with ends $v \in V$ and $u \in U$ may also be represented as (v, u). Degree of a vertex is the number of edges incident to it, and $N(v)$ represents the set of neighbors of v, that is, the set of vertices with an edge connecting them to v.

Walk, path, and cycle: A walk in a graph is a sequence of edges in which each edge shares an end with the next edge in the sequence. You can view this as a movement that starts from a vertex and continues by taking edges and visiting vertices on the way. A path is a walk that visits each vertex at most once. A cycle is just like a path, except that it starts and ends in the same vertex. We say that a graph $G(V, E)$ is connected if for every $v, u \in V$ there is a path connecting v and u.

Forest and tree: A graph $H(V', E')$ is a subgraph of a graph $G(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A connected component of a graph G is a maximal connected subgraph of G. A forest is a graph that has no cycles. A tree is a connected forest, and so any connected component of a forest is a tree. A useful property of trees is that for each pair of vertices u and v there is a unique uv-path connecting them.

Rooted tree: When we say a tree t is rooted at v, we are assigning depth 0 to v and depth i to any vertex of distance i from v. Since there is a unique path from the root to each vertex, any vertex at depth $0 < i$ has exactly one edge to a vertex in depth $i-1$, and the other end of that edge is referred to as the parent of

this vertex. The rest of the neighbors of this vertex are referred to as the children of it, and they all lie at depth $i + 1$. Then ancestors of a vertex are defined as its parent and the ancestors of the parent, and the desendants of a vertex are its children and the desendants of its children. It should be noted that each edge in a rooted tree connects a child to its parent.

Depth-first traversal: Suppose we have a rooted tree T and we wish to traverse this tree starting from the root and by walking on its edges. One useful way to do this is the depth-first search: When at a vertex v, traverse the edge to a yet unvisited child of v, and if v has no unvisited children, return to the parent of v. Induction immediately verifies that after a vertex v is visited, all of its descendants are visited before returning to the parent of v. If v has no unvisited children and no parent, then you are at the root and all vertices are visited, therefore the traversal is terminated. We use depth-first ordering to refer to the order in which vertices are visited in a depth-first traversal of the tree.

One nice property of the depth-first traversal is that each edge is only traversed twice: once from parent to child and once from child back to parent. We will use this fact in later chapters.

Weighted graph: The edges of a graph may have weights associated with them. If c_e is the weight of the edge e in $G(V, E)$, then for a subgraph H we use $w(H)$ to refer to the total weight of the edges in H. In a graph $G(V, E)$ if for any $u, v, w \in V$ we have $c_{u,v} + c_{v,w} \leq c_{u,w}$, then G is a metric graph. It is easy to see that for any st-path P in a metric graph, it must be that $c_{(s,t)} \leq w(P)$.

Bipartite graph: A graph $G(V, E)$ is bipartite if V can be partitioned into two sets X and Y such that there is no edge with both ends in one of X and Y.

2.2 Linear Programming

Suppose that a problem can be formulated by n non-negative variables and a series of m linear constraints of the form $\sum_{i=1}^{n} a_{ij}x_i \geq b_j$ for each $1 \leq j \leq m$ where a_{ij} and b_j are constants, and with an objective function of the form $\sum_{i=1}^{n} c_i x_i$. Such a formulation is referred to as a linear program (LP) and can the optimal solution for it can be found in polynomial time.

Integer programming: In most combinatorial problems we are seeking discrete solutions with integer or binary variables, and linear programs cannot necessarily capture such a constraints. As such, solving a combinatorial problem using linear programming usually consists of modeling the problem as a linear program but with the assumption that variables can be constrained to be integers. This is referred to as an integer relaxation (or integer program) of the problem. Then the integrality constraints are removed so that a solution to the linear program can be found in polynomial time, yielding a solution that may consist of fractional values for our variables. Then we would need to "round" the fractional solution to a valid solution to our original discrete problem.

LP rounding Since removing the integrality constraint relaxes the problem, the fractional solution produced by the linear program is of no worse quality than the optimal solution for the problem, that is if our original integer relaxation was indeed a mathematical formulation of the problem. Therefore, we can start from the fractional solution to the linear program and use its cost as a lowerbound on the optimal cost. Then in the rounding procedure if the cost increases by a factor of α, we can claim our algorithm is an α approximation for the problem.

Integrality gap: If the best solution to the integer relaxation is large relative to the fractional solution of the LP, then one cannot hope for an LP rounding algorithm with small approximation guarantee. More precisely, if the ratio of the best solution of the integer relaxation to the best solution of the corresponding linear relaxation is β, then there can be no such a rounding algorithm based on this relaxation with an approximation factor better than β. This ratio is known as the integrality gap of the relaxation.

Primal-dual approach: The dual of a (primal) relaxation described above is a formulation by m non-negative variables and a series of n linear constraints of the form $\sum_{j=1}^{m} a_{ij} y_j \geq c_i$ for each $1 \leq i \leq n$, and with a maximization objective function of the form $\sum_{j=1}^{m} b_j y_j$. A key property that ties the primal and dual relaxations is that their objectives are equal for their respective optimum solution. This implies that any solution for the dual has an smaller or equal objective than any solution to the primal. Therefore, one can use any solution of the dual to establish a lowerbound on the objective value of the optimal solution of the original integer relaxation. Such a lowerbound then can be used to prove approximation guarantees. This technique is known as primal-dual.

2.3 Warm-up problem

The problems that we are concerned with in this thesis all can be placed under the class of service-demand problems. In each such a problem, we have a set $\mathcal{D}$ of demands and a set $\mathcal{F}$ of facilities and the goal is to assign to each demand point $i \in \mathcal{D}$ a facility in order to minimize some cost function. Each problem might have some other constraints, for example, in Chapter 3 the demands and

facilities have certain types that correspond to vertices of a given type tree, and a demand of type x can be only satisfied with facilities that are either of type x or of a type that is an ancestor of type x.

For now let us focus on a simple problem: a set $\mathcal{D}$ of demands and a set $\mathcal{F}$ of facilities are given along with costs c_{ij} for each $i \in \mathcal{D}$ and $j \in \mathcal{F}$ where c_{if} represents the cost of matching i with f. The goal is to satisfy the demands with the smallest cost, that is, to mach each demand point $i \in \mathcal{D}$ with one facility such that the total cost of matched pairs is minimized. We can formulate this as an integer program using a variable x_{ij} that is defined for each $i \in \mathcal{D}$ and $j \in \mathcal{F}$ such that x_{ij} is 1 if i is matched with j and it is 0 otherwise. With this, observer that Formulation Example IP is an integer relaxation of this simple demand-satisfaction problem.

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{D}} \sum_{f \in \mathcal{F}} c_{if} x_{if} && \text{(Example IP)} \\ \text{s.t.} \quad & \sum_{f \in \mathcal{F}} x_{if} = 1 \quad \forall i \in \mathcal{D} \\ & x_{if} \in \{0, 1\} \quad \forall i \in \mathcal{D}, f \in \mathcal{F} \end{aligned}$$

To solve this problem exactly, for each demand $i \in \mathcal{D}$ we can find the facility $f \in \mathcal{F}$ with the smallest c_{if} and assign i to f. This would satisfy all of the demands while minimizing the total cost. For each $i \in \mathcal{D}$, the facility $f \in \mathcal{F}$ that minimizes c_{if} can be found in $O(|\mathcal{F}|)$ iterations, and therefore the total running time for this algorithm is $O(|\mathcal{D}||\mathcal{F}|)$ which happens to match the size of the input.

CHAPTER 3

A LOT-SIZING PROBLEM

The *facility location problem* consists of a set of clients and a set facilities with given distances between each facility and each client. There is a cost for opening each facility, and the question is which facilities to open in order to serve the clients most cost-efficiently. The cost of a solution is the sum of opening costs for the facilities that are chosen to be opened plus the total cost of serving all of the clients, where the cost of serving each client is equal to its distance from the closest opened facility.

Although algorithm design for the facility location problem and its many variants has been the focus of a significant body of research, there is an important class of generalizations that has received much attention, but only with more limited success, that is, facility location with types.

Simplistic models of facility location problem place no restrictions on the clients served by each facility, which is typically an unreasonable assumption, and so a great deal of work has been done to address further constrained models. The simplest extension is to impose a capacity constraint, but for a wide range of applications it is more appropriate to distinguish types of service provided and required, and in this chapter it is on this class of problems that we shall focus. Facility location models are closely linked to a number of inventory models, since one can view a facility as a point in time at which an order is placed, and the assignment cost then corresponds to the cost of filling a particular demand from that order, where one can have fixed ordering costs, unit ordering costs, and inventory holding costs easily incorporated into this setting. Here we consider an inventory management problem that arises in the setting

with type constraints.

Since the facility location problem is NP-hard, much of the algorithmic work in this domain has focused on the design of approximation algorithms, and it has proved to be a fertile ground for the development of many of the now-standard techniques in algorithm design: researchers have applied deterministic and randomized rounding [30, 24, 10], primal-dual methods [17], local search[18], and even greedy-type algorithms [16] to this problem. For the classical facility location problem, the best approximation guarantee currently known is a 1.488-approximation algorithm (where a *ρ-approximation algorithm* is a polynomial-time algorithm that is guaranteed to find a feasible solution of objective function value within a factor of ρ of the optimum), which was obtained by randomized rounding [24]. For the *k-median problem*, in which there are no facility costs but one is limited to opening only k facilities, strong results are known via a range of techniques as well [25, 17, 2], and analogous results (though more limited) are known for the capacitated version of the facility location problem [11, 1, 28, 4]. One particularly notable open problem is to derive good algorithms for the setting in which the opening/ordering costs are submodular set functions of the set of demand points assigned (or for suitably general special cases).

The research presented in this chapter aims to derive results in which each facility may have a different serving capability, particularly, *Facility Location with Hierarchical Types*. In this problem, the types form a rooted tree in which a facility of a given type can serve any client that has a type that is a descendant of the facility type in the tree (where a node is trivially a descendant of itself).

Analogous to the development of approximation algorithms for facility location problems, there has been a corresponding thread of research investigating results of a variety of inventory management problems, primarily the *lot-sizing* and the *joint replenishment problem*[19, 20, 21, 6, 9, 23]. In some elements, these problems are sometimes simpler than their facility location equivalents, since the assignment costs, in addition to obeying the triangle inequality, often have an even more refined structure due to the linear nature of ordering/demand time periods. However, these problems have a harder element, since the metric is definitely not symmetric - one cannot serve a demand point earlier in time than the ordering period. Hence, it is natural to ask for the analogs of the two results discussed above. Surprisingly, for the lot-sizing problem with hierarchical types, we show that the problem can be solved dynamic programming in polynomial time, in a manner analogous to a recent result [9]. However, unlike for the unconstrained lot-sizing problem for which the natural linear programming relaxation is the basis for a primal-dual polynomial-time algorithm as well [21], we give an example for which there is an integrality gap of 4/3 for the lot-sizing problem with hierarchical types.

3.1 Notation

In this section, we introduce some definitions and notation to be used throughout this Chapter. Let $\mathcal{F}$ be the set of potential facilities; opening facility $i \in \mathcal{F}$ has associated non-negative cost f_i. Let $\mathcal{D}$ be the set of demand points. We need to assign each client $j \in \mathcal{D}$ to some open facility i and it costs c_{ij}. In addition, there is set of types $\mathcal{T}$, forming a partial ordering. Each facility i has a type $t(i)$ and each client j has type $t(j)$. Facility i is capable of serving client j if and

only if $t(j) \leq t(i)$ in the partial ordering. Types indicate the serving capabilities. Denote $\mathcal{F}(j)$ be all facilities that can serve client j.

Suppose we open the set of facilities S in the solution and assign client j to $\sigma(j)$. Then the total cost of this solution is $\sum_{i \in S} f_i + \sum_{j \in \mathcal{D}} c_{\sigma(j)j}$, i.e., the sum of total facility opening cost and total assignment cost. The goal is to find a feasible solution with minimum total cost.

Our algorithm and analysis will rely on the following primal linear-programming (LP) relaxation P and its dual D.

$$\begin{aligned} \min \quad & \sum_{i \in \mathcal{F}} f_i y_i + \sum_{j \in \mathcal{D}} \sum_{i \in \mathcal{F}(j)} c_{ij} x_{ij} && \text{(P)} \\ \text{s.t.} \quad & \sum_{i \in \mathcal{F}(j)} x_{ij} \geq 1 \quad \forall j \in \mathcal{D} \\ & x_{ij} \leq y_i \quad \forall j \in \mathcal{D}, i \in \mathcal{F}(j) \\ & x_{ij}, y_i \geq 0 \quad \forall i \in \mathcal{F}, j \in \mathcal{D} \end{aligned} \qquad \begin{aligned} \max \quad & \sum_{j \in \mathcal{D}} v_j && \text{(D)} \\ \text{s.t.} \quad & v_j \leq c_{ij} + w_{ij} \quad \forall j \in \mathcal{D}, i \in \mathcal{F}(j) \\ & \sum_{j : i \in \mathcal{F}(j)} w_{ij} \leq f_i \quad \forall i \in \mathcal{F} \\ & v_j, w_{ij} \geq 0 \quad \forall j \in \mathcal{D}, i \in \mathcal{F}(j). \end{aligned}$$

Any binary feasible solution to (P) corresponds to a feasible solution. $y_i = 1$ indicates that facility i is open and $x_{ij} = 1$ indicates that client j is served by facility i.

For the dual (D), we can intuitively think v_j as budget of client j and w_{ij} as its contribution towards opening facility i. The first set of constraints say that the budget can cover both assignment cost and contribution toward opening cost. The second set of constraints say that the total contribution toward one facility cannot exceed its opening cost.

3.2 Lot-sizing with hierarchical types

In this section, we consider a related problem where facilities and clients are embedded in time, and thus may correspond to orders and demands occurring over time. Let order i occur at time $\tau(i)$ and demand j occur at time $\tau(j)$. We still have the hierarchical type tree as in *facility location with hierarchical types*. For a client j to be served by facility i, in addition to the type constraint $t(j) \leq t(i)$, we also have the time constraint, $\tau(i) \leq \tau(j)$. For the assignment cost c_{ij}, we have a non-decreasing property: for two facilities i, i' and client j with $\tau(i) \leq \tau(i') \leq \tau(j)$, we have $c_{ij} \geq c_{i'j}$. Also, $c_{ij} = \infty$ if $\tau(i) > \tau(j)$.

3.2.1 A dynamic programming algorithm

We now describe a polynomial-time dynamic programming (DP) formulation to solve the lot-sizing problem with hierarchical types. For this, we need some new notation. Considering a type tree $\mathcal{T}$, we define $\mathcal{T}(t)$ to be the subtree of $\mathcal{T}$ rooted at t if $t \in \mathcal{T}$, and $\emptyset$ otherwise. We also use $C_{\mathcal{T}}(t)$ to denote the set of children of t in $\mathcal{T}$. For convenience we add a dummy facility χ of the root type occurring at $\tau(\chi) = -\infty$ with opening cost $f_\chi = 0$ such that $c_{\chi j} = \infty$ for any client j.

Now, for any facility i, any type t satisfying $t \leq t(i)$, and any time $\tau > \tau(i)$ we define $\mathcal{L}(i, \mathcal{T}(t), \tau)$ to be the cost of the optimal solution for the subset of the clients and facilities contained in the interval $[\tau(i), \tau)$ that have types contained in the subtree rooted at t, i.e., $\mathcal{T}(t)$, with the extra assumption that i has opening cost $f_i = 0$. Thus, the optimal overall cost of the lot-sizing problem is $\mathcal{L}(\chi, \mathcal{T}, \infty)$.

Note that there are at most $|\mathcal{D}|$ values for the third element that are of interest, and therefore there are a total of $O(|\mathcal{F}||\mathcal{T}||\mathcal{D}|)$ distinct sub-problems.

The base case of the DP is when the type tree has depth zero, and therefore the optimal solution has cost 0. Let us calculate $\mathcal{L}$ using a recursive formula that uses induction on the depth of the type tree, each time finding the latest facility of the root type that is opened in the optimal solution. For this, we can establish the following recurrence.

$$\mathcal{L}(i, \mathcal{T}(t), \tau) = \min \Big\{ \sum_{j:t(j)=t, \tau(i) \leq \tau(j) < \tau} c_{ij} + \sum_{t' \in C_{\mathcal{T}}(t)} \mathcal{L}(i, \mathcal{T}(t'), \tau),$$
$$\min_{i':t(i')=t, \tau(i)<\tau(i')<\tau} \{ f_{i'} + \mathcal{L}(i, \mathcal{T}(t), \tau(i')) + \sum_{t' \in C_{\mathcal{T}}(t)} \mathcal{L}(\tau(i'), \mathcal{T}(t'), \tau) \} \Big\}$$

The first case of the outer min corresponds to the scenario when no facilities of type t are opened in the optimal solution in the interval $[\tau(i), \tau)$, other than i. Here, all clients of type t will be served by i, and the rest of the solution can be separately calculated for each of the type trees $T(t')$ for all $t' \in C_{\mathcal{T}}(t)$, and be summed to get the total cost.

The second case considers the facility i' that is the last facility of type t opened after i and by time τ. Here, the optimal solution for the interval $[\tau(i'), \tau)$ is calculated as in the previous case, and the optimal solution for the interval $[\tau(i), \tau(i')]$ is calculated by recursing on $\mathcal{L}(\tau(i'), \mathcal{T}(t'), \tau)$.

Together, the two cases cover all possibilities for the latest facility of type t that is opened in the time interval $[\tau(i), \tau)$ and therefore at some point the optimal solution is considered. Each update can be implemented in $O(|\mathcal{F}| + |\mathcal{D}|)$ given that the facilites and clients of each type are maintained sorted by

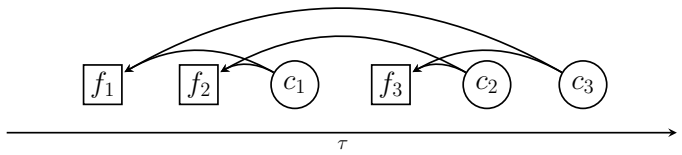

Figure 3.1: The instance of the lot-sizing problem inducing integrality gap of $4/3$ for LP and the corresponding support of x. All drawn arcs correspond to half integral values.

occurence time, and therefore the total runtime is bounded by $O(|\mathcal{F}|^2|\mathcal{T}||\mathcal{D}| + |\mathcal{F}||\mathcal{T}||\mathcal{D}|^2)$.

Theorem 1. *There exists a polynomial-time dynamic programming algorithm to solve the hierarchical lot-sizing problem.*

3.2.2 An integrality gap example

Interestingly, although this problem admits polynomial solution, we can provide an example where the natural linear program has an integrability gap of $\frac{3}{4}$. We need only two types, say a and b, where a is the root type and b is the only leaf type.

The instance consists of three facilities $f_1, f_2,$ and f_3 and three clients $c_1, c_2,$ and c_3 such that $\tau(f_1) < \tau(f_2) < \tau(c_1) < \tau(f_3) < \tau(c_2) < \tau(c_3)$. Also, facilities f_1 and f_3 and the client c_3 are of type a and the rest of clients and facilities are of type b. Let us say the opening cost for all facilities is a constant c and the holding costs are 0, except for $c_{f_1 c_2} = \infty$. It is easy to verify that these holding costs are non-decreasing.

Any solution that opens two of the facilities will have a total cost of $2c$ and would be optimal. However, in a fractional solution one can open each facility

halfway since each client can be served by two facilities without paying a holding cost, i.e., $y_{f_1} = y_{f_2} = y_{f_3} = 1/2$. See Figure 3.1 for an illustration. Such a fractional solution satisfies all constraints of LP and has a cost of $3\frac{c}{2}$, inducing an integrality gap of $\frac{4}{3}$.

Theorem 2. *The natural LP for lot sizing has integrality gap* $\geq \frac{4}{3}$.

3.2.3 Upperbound on Gap

Here we provide a 2-approximation algorithm that bounds the integrality gap of the LP for lot sizing with hierarchical types. Consider an instance $\mathcal{I}$ of the problem and let us start with an optimal dual solution (v^*, w^*).

To obtain an integral solution we iterate over the types in $\mathcal{T}$, starting from the root type r and ending with leaves, each time focusing on a type t and opening facilities in order to serve all clients of type t. We also need to maintain a dual solution (v^+, w^+) for the set of clients that are not served yet. We initialize $(v^+, w^+) = (v^*, w^*)$, and modify the given instance by setting $c_{ij} = \infty$ for any client j and facility i such that $\tau(j) - \tau(i) > v^*_j$.

Since in for each type the algorithm at some iteration opens facilities to serve all clients of that type, in the end all clients will be served by S. Next theorem bounds the cost of the obtained solution.

Theorem 3. *The cost of the solution produced by this algorithm is at most* $2\sum_j v^*_j$.

Proof. Each time a facility i is opened serving a set S_i of previously unserved clients, a total cost of $\sum_{j\in S_i} v^+_j$ is induced to the integral solution as $\sum_{j\in S_i} v^+_j = \sum_{j\in S_i} w^+_{ij} + \sum_{j\in S_i} c_{ij} = f_i + + \sum_{j\in S_i} c_{ij}$. Since v^+_j freezes when j is served and

Algorithm 1 Finds an integral solution of cost at most twice the LP objective.

INPUT: An instance $\mathcal{I}$ on a type tree $\mathcal{T}$ and a maximum dual solution (v^*, w^*). **OUTPUT:** A set S of facilities that will be opened.

```
I' ← I
(v+, w+) ← (v*, w*)
S ← ∅
for all types t ∈ T, starting from root and traversing towards leaves do
  ⟨Facility opening phase⟩:
  while there are clients of type t in I' that are not yet served by S_t do
    Consider the first unserved client i of type t.
    Add to S_t the tentatively opened facilities that can potentially serve i and
    appears last.
  end while
  ⟨Pruning phase⟩:
  for all clients j in I' that are served by S_t do
    freeze v_j^+
    remove j from I'
  end for
  set the cost of facilities in S_t to 0
  ⟨Dual update phase⟩:
  for all clients j of type t(j) ≤ t, starting from leaves and traversing towards
  t do
    increase v_j^+ while dual remains feasible in I'.
    while exists a client j' and ε > 0 where (j, j', ε)-tradeoff is possible do
      v_{j'}^+ ← v_{j'}^+ + ε
      v_j^+ ← v_j^+ − ε
    end while
  end for
  S ← S ∪ S_t
end for
```

each client contributes to at most one such facility, the final cost of the constructed solution would be at most $\sum_j v_j^+$. Next we show that $\sum_j v_j^+ \leq 2 \sum_j v_j^*$.

Consider the set S_t of facilities that are opened when the outer loop iterates over type t, and let $S'_t = \{j \in S_t :\}$ be the subset of S_t containing any clients that satisfies $v_j^+ = v_j^*$. Then we claim that in the dual update phase of the algorithm for type t the total increase in the objective function is at most $\sum_{S'_t} v_j^*$. For this,

consider a client j of type t whose dual variable increases in dual update phase of some type t', that is, $v_j^+ > v_j^*$ when the algorithm focuses on type t. Then we claim that when j is removed, the objective function will not increase. The reason is that when v_j^+ is being increased, any client j' of a lower type is already bound by a tentatively opened facility i. Since we are able to increase v_j^+, it must be that j cannot be served by i. Therefore, any increase in the dual objective must stem from removal of a client that has not had its dual variable increased.

Also, since we started with an optimal dual solution, we know that for any client j in S' there is at most one client whose dual variable can be increased when the dual variable of j, v_j^*, is decreased. This property is maintained during the dual update phase, and therefore when we remove a client j, there will be at most one client whose corresponding dual variable will be increased in the dual update phase, and the increment is no more than $v_j^+ = v_j^*$.

By these arguments, we can say that the total increment in the dual objective in the dual update phase is no more than the total dual value of any clients j that were served in the facility opening phase with $v_j^+ = v_j^*$. Therefore we have $\sum_j v_j^+ - v_j^* \leq \sum v_j^*$ which concludes the lemma. □

CHAPTER 4

AN EXAM-SCHEDULING PROBLEM

Here we review a problem arising when scheduling final exams for Cornell University. Many of the actual challenges are relaxed here for the sake of simplicity.

4.1 Background

Every semester at Cornell University about 20 thousand students take exams for about 700 different courses at the end of each semester in a period of roughly one week. Each day there are at most 3 different times where exams can be scheduled (usually 9am, 2pm and 7pm), and in total during the exam period there are about 21 available time-slots when exams are held. The task of assigning each exam to a time-slots is a complex scheduling problem that has been traditionally handled via a combination of a "natural hack" and computer assistance.

The natural hack relies on the assumption that two courses with overlapping meeting times do not share registered students. This means that if we color courses based on their meeting times such that all courses of the same color have overlapping meeting times, then assigning all exams corresponding to the courses of the same color to one time-slot should not produce any conflicts.

A limitation of this "hacking" approach is that it cannot be applied to courses that are offered with multiple meeting-time options but a single final exam. For example, of the 689 exam-items that needed to be scheduled for Spring 2016, 69 corresponded to courses with multiple meeting times. This is were we entered

the scene: The registrar was able to color the rest of the 620 courses with 14 colors and we were asked to use computer assistance to color the remaining 69 courses with 6 colors. The objective was to do this with introducing the smallest number of conflicts, that is, students with two exams of the same color. While we were able to achieve 0-conflict coloring for our part, it turned out that the coloring produced by the hack included 83 conflicts.

Motivated by the initial success of our algorithm for the smaller set of exams, we took on the task of coloring and scheduling all of the 689 exams using computer assistance. Our approach is in two phases: first we focus on finding a coloring of the courses that assigns to each course one of 20 colors such that there are no students enrolled in two courses of the same color. Then in the second phase we assign each of the 20 colors to one of the 20 time-slots while minimizing undesirable instances such as back-to-back exams and cases where a student has to take three exams in a period of 24 hours. Using this approach we were able to produce a scheduling of exams within the given time frame without any direct conflicts.

Upon further investigation and discussion with the client it became clear that the true objective is the total number of instances of conflicts, plus the number of instances of three exams in 24 hours. This new objective provided us with a challenge since our two phase approach gives strict priority to minimizing the number of conflicts, and treats the number of cases with three exams in 24 hours as the secondary objective. In this chapter we discuss a way to go around this issue by changing the two phase structure, and focus on a theoretical problem that arises when so.

4.2 Formulations

Each of the two phases of our approach is handled via a mixed integer formulation. Here we review each of these formulations in a simplified environment.

4.2.1 Phase 1: The Coloring Problem

The integer relaxation for phase 1 captures the graph coloring problem. Given a graph G, can we color the vertices with k colors such that the ends of each edge have different colors? The minimum number of colors required to color a graph is known as the *chromatic number* of that graph. Theoretically, this problem is extremely difficult: Even when the given graph is 3-colorable, although there has been an extensive effort, no algorithm with a factor better than $O(n^{\epsilon})$ is known.

Let $\mathcal{F}$ be the set of colors available to us, and $\mathcal{D}$ be the set of exam items that need to be colored. Also for each pair $i, j \in \mathcal{D}$ let c_{ij} be the number of students that would need to take both exams i and j, which corresponds to the overhead on the total number of conflicts if i and j are assigned the same color. Then Coloring IP is an integer relaxation for the problem of finding a coloring of the exams with $|\mathcal{F}|$ colors such that the number of conflicts is minimized.

	18 colors	19 colors	20 colors
number of conflicts	7	3	0

Table 4.1: The results of the coloring phase for the data regarding Spring of 2016.

$$\begin{aligned}
\min \quad & \sum_{i,j \in \mathcal{D}} c_{ij} h_{ij} && \text{(Coloring IP)} \\
\text{s.t.} \quad & \sum_{f \in \mathcal{F}} x_{if} \geq 1 \quad \forall i \in \mathcal{D} \\
& x_{if} + x_{jf} \leq 1 + h_{ij} \quad \forall i, j \in \mathcal{D}, f \in \mathcal{F} \\
& x_{if} \in \{0, 1\} \quad \forall f \in \mathcal{F}, i \in \mathcal{D} \\
& h_{ij} \in \{0, 1\} \quad \forall i, j \in \mathcal{F}
\end{aligned}$$

x_{if} in Coloring IP for an exam i and color f is a binary variable that indicates that i is of the color f. If two exams $i, j \in \mathcal{D}$ are assigned the same color f, then h_{ij} has to be 1 which causes c_{ij} many conflicts to be counted towards the objective. The first constraint ensures that each exam is assigned a color, and therefore this relaxation captures the problem of minimizing the total number of conflicts.

With this relaxation and for the case of Cornell University a conflict-free coloring with 20 colors can be produced using Gurobi as the mixed integer solver in a matter of hours.

4.2.2 Phase 2: Scheduling Problem

In this phase, already having a conflict-free coloring, we focus on assigning each color to a time-slot in order to minimize a cost function that is a combination of the number of instances of *3 exams in 24 hours* and number of *beck-to-back exams* that the students have to take.

Suppose we only wish to minimize the number of instances of back-to-back exams that the students take, and consider the graph where each vertex represents a color and the weight of edges represent shared students between colors. The schedule starts from a vertex, and places all of the vertices in a total order. For each pair of vertices u and v that are scheduled in two consecutive time-slots, there are as many back-to-back cases as the weight of the (u, v) edge. Therefore, we can see the schedule as a path through this graph that visits all of the vertices, and the goal is to minimize the total weight of the edges of this path. This problem is known as the Traveling Salesman Problem, and it is NP-hard even for metric graphs.

Let $\mathcal{F}$ be the set of time-slots and $\mathcal{D}$ be the set of colors that are to be scheduled in these time-slots. For each pair $i, j \in \mathcal{D}$ let c_{ij} represent the number of students that have an exam in both colors. For each time-slot $f \in \mathcal{F}$ we use $n(f)$ to refer to the next time-slot, and if f is the last time-slot let $n(f)$ be the first time-slot. For a time-slot $f \in \mathcal{F}$ let c_{ijkf} be the cost induced by the three exams i, j, and k if they are scheduled at $f, n(f)$, and $n(n(f))$, respectively.

$$\min \quad \sum_{i,j,k\in\mathcal{D}} \sum_{f\in\mathcal{F}} c_{ijkf} x_{ijkf}$$

(Scheduling IP)

$$\text{s.t.} \quad \sum_{k\in\mathcal{D}} x_{kijf} = \sum_{k\in\mathcal{D}} x_{ijkn(f)} \quad \forall i,j \in \mathcal{D}, f \in \mathcal{F}$$

$$\sum_{i,j,k\in\mathcal{D}} x_{ijkf} = 1 \quad \forall f \in \mathcal{F}$$

$$\sum_{j,k\in\mathcal{D}} \sum_{f\in\mathcal{F}} x_{ijkf} = 1 \quad \forall i \in \mathcal{D}$$

$$x_{ijkf} \in \{0,1\} \quad \forall i,j,k \in \mathcal{D}, f \in \mathcal{F}$$

x_{ijkf} in Scheduling IP indicates that i is scheduled for time f while j and k are scheduled for $n(f)$ and $n(n(f))$, respectively. The first constraint guarantees consistence juxtaposition of exams i and j, while the second and third constraints make sure that each color is assigned to one time-slot and each time-slot has exactly one color assigned to it. Together, these constraints ensure that x establishes a total order upon the colors in any feasible solution.

Although this formulation has n^4 variables, since n is small we can use Gurobi to find the optimal solution in less than an hour. Our cost function c_{ijkf} captures a weighted combination of the number instances of three exams in 24 hours and back-to-back exams induced by having i, j and k scheduled consecutively starting from time-slot f, along with some other undesirable instances.

4.3 Path cover problem

One problem with this approach is that the color of each course is fixed in the first phase without any regard for the other part of the objective, that is, the number of instances of three exams in 24 hours. Attempting to incorporate this objective into the integer relaxation of phase 1 makes the solving of the problem by Gurobi painstakingly slow and practically infeasible.

To remedy this we add a phase 1.5 that "prepares" the conflict-free coloring for the scheduling. We do this by finding chains of colors that could be potentially scheduled sequentially in the final schedule and focus on improving the coloring relative to these chains while maintaining the coloring conflict-free.

In this chapter we are mainly concerned with an algorithm for finding these chains. Consider the graph where each color is represented with a vertex and the weight of an edge (u, v) is the number of students that are enrolled in a course of color u and a course of color v. Notice that by the nature of our coloring each student is enrolled in at most one course of each color, and therefore is only counted once towards the weight of an edge. If vertices v and u are scheduled consecutively in phase 2, then they would contribute to the total number of instances back-to-back exams as much as is the weight of (u, v). The algorithm in phase 2 attempts to also minimize the number of back-to-back exams, and therefore if an edge has small weight it is more likely for the corresponding pair of vertices to be scheduled consecuitively in phase 2.

Our algorithm in phase 1.5 takes such a graph as input along with a subset of its edges and tries to modify the coloring in order to minimize the number of students with exams on both ends of the given edges while maintaining the

conflict-freeness of the coloring. Now the problem is which edges should be selected for the input of this phase. For this we wish to select a set of edges in the given graph with minimum total weight such that these edges form a set of disjoint paths in the graph. This way we can hope that each selected path is likely to be scheduled in the same order in phase 2, therefore minimizing the instances of students with exams on both ends of the edges of these paths translates to a reduction of the number of instances of back-to-back exams that students must take in the final schedule, which would also reduce the number of instances of students with 3 exams in 24 hours.

One way to go about this is to find a path of $n-1$ edges of minimum total length. This would be equivalent to the Traveling Salesman Path Problem (TSPP) which is a well-studied problem. However, such a path is unlikely to be scheduled in order in phase 2 as our main objective in phase 2 is to reduce the number of instances of 3 exams in 24 hours and not the number of instances of back-to-back exams. Therefore some of the optimization effort in phase 1.5 would be wasted on edges that will not be scheduled consecutively.

Instead of just focusing on a path of length $n-1$, we consider all numbers $1 \leq k \leq n-1$ and for each such k we seek to find a set of disjoint paths covering a total of k edges with minimum total weight, and then each time we try the optimization of phase 1.5 on resulting set of k edges, and for each produced "modified" coloring we run the scheduling of phase 2. The best solution for all k would be the output of our exam-scheduling algorithm. When $k = n-1$, this would be equivalent to solving the TSPP, and therefore, our problem here is a generalization of TSPP. TSPP is inapproximable in general graphs, and therefore we focus on metric graphs.

We define *k-edge path-cover* to be a set of disjoint paths covering exactly k-edges, and in this chapter we present a 2-approximation for the problem of finding the minimum weight k-edge path-cover in a given metric graph.

4.4 the k-edge path-cover problem

Our approach for the k-edge path-cover problem is relatively standard: First we find a forest of k edges with minimum weight, and then we turn each tree into a path by at most doubling its weight.

Algorithm 2 Finds a k-edge path cover with a total weight at most twice that of the optimal solution.

INPUT: A metric graph $G(V, E)$. **OUTPUT:** A set S of disjoint paths covering a total of k edges.

```
T ← ∅
while |T| ≤ k do
  consider the next (u, v) ∈ E in order of weight, from lightest to heaviest.
  if u and v belong to different connected components of T then
    T ← T ∪ (u, v)
  end if
end while
S ← ∅
for all connected component t in T do
  Let p be the path corresponding to the depth-first traversal order of t from
  an arbitrary root.
  S ← S ∪ p
end for
```

Theorem 4. *Algorithm 2 is a 2-approximation algorithm for the k-edge path-cover problem.*

Proof. First we show that the cost of the forest T is no greater than $w(\mathrm{OPT})$. Note that OPT is also a forest, and we generalize the proof to show that T is the cheapest forest among forests in G with k edges. Let OPT^{-} be the forest

that results from removing the heaviest edge of OPT. Similarly, let T^- be the forest that results form removing the heaviest edge of T. By induction on k we have $w(T^-) \leq w(\mathrm{OPT}^-)$. Let e_T and e_{OPT} be the heaviest edge in T and OPT respectively. If $w(e_T) \leq w(e_{\mathrm{OPT}})$ we are done. For the sake of contradiction suppose $w(e_T) > w(e_{\mathrm{OPT}})$, and notice that in this case all edges of OPT must be lighter than $w(e_T)$. However, in such a case there must be some edge in OPT with at least one end not spanned by T^-, and that edge would have priority over e_T in construction of T in Algorithm 2, which is a contradiction.

In Chapter 2 we argued that in the depth-first traversal of a tree each edge is traversed exactly twice. Therefore if we consider the depth-first ordering of a metric tree t, then the weight of an edge that connects a consecutive pair of vertices in this ordering is no more than the weight of the traversed path between the corresponding visits in the depth-first traversal. Since in the whole process each edge is traversed twice, therefore, the total weight of the path corresponding to the dpeht-first ordering of the tree is at most twice the weight of the whole tree.

Since each path weights at most twice its corresponding tree, the total cost of all of the paths is at most twice the weight of the forest, therefore $w(S) \leq 2w(T) \leq w(\mathrm{OPT})$ $\square$

CHAPTER 5

A CLUSTERING PROBLEM

5.1 Introduction

Clustering problems have been studied from a range of algorithmic perspectives, due to the breadth of application domains for which they serve as appropriate optimization models. This is particularly true from the context of approximation algorithms, in which a wide variety of these optimization models have been shown to have polynomial-time algorithms with strong performance guarantees. In this chapter, we will consider an important generalization of many of these problems for which no constant performance guarantee had been known previously, and give the first constant performance guarantee for a central special case that is a common generalization of two of the most well-studied models, the k-center and k-median problems.

In the metric k-supplier problem, we are given a set $\mathcal{D}$ of n demand points and a set $\mathcal{F}$ of m service points in a common metric space. The goal is to open k of these service points in order to serve the demand in $\mathcal{D}$, while minimizing a connection cost. Typically, when there are no capacities on how many demand points that each service point can serve, each demand point is served by its closest opened service point. Although in this chapter we only focus on cases where $\mathcal{F} = \mathcal{D}$, for the sake of readability we try to distinguish between the two roles that any of the given points may play at any point.

We use $c_{jj'}$ denote the distance between points j and j'. Similarly, for sets S and D of points and a point $j \in D$, c_{jS} refers to the distance between j and a

point in S that is closest to it, and c_{DS} is the array of length $|D|$ of non-decreasing distances c_{jS} for each $j \in D$. For a solution S with $|S| \leq k$, the connection cost of the k-supplier problem can be expressed as function of $c_{\mathcal{D}S}$.

In the *ordered median problem*, we are given a weight vector w, where $w(i)$ for $i \in [n]$ is the weight on the ith longest connection, and the cost of a solution $S \subset F$ with $|S| \leq k$ is simply $w^T c_{DS}$. Very recently, Aouad and Segev [3] presented an $O(\log n)$ approximation algorithm for the case when weights in w are non-decreasing. This result intensifies the question of finding conditions for w under which the ordered median problem can be approximated with a constant quality guarantee.

For example, again when $\mathcal{F} = \mathcal{D}$, the ordered median problem captures the k-center problem, by setting the cost vector w to be 0s for each index, except for the last index $w(n) > 0$. So the problem here is to select a set $S \subset \mathcal{F}$ of k centers in order to minimize the maximum distance in $c_{\mathcal{D}S}$. The first approximation algorithms for this problem were presented by Hochbaum & Shmoys [14] and Gonzalez [12], with an approximation factor of 2. The latter algorithm starts with $S = \{i\}$ for some arbitrary $i \in \mathcal{F}$, and in each iteration adds the point $i' \in \mathcal{F}$ that maximizes $c_{i'S}$ to S, provided $|S| < k$. Suppose $j \in \mathcal{D}$ is the furthest demand point from S at the end of this algorithm. Then, if you consider the balls of radius $c_{jS}/2$ around each point in S, these balls must be disjoint, or else j should have been added to S at some point by the algorithm. If we assume that there is a set S^* with $\max(c_{\mathcal{D}S^*}) < c_{jS}/2$, then the ball of radius $c_{jS}/2$ centered at each of $i \in S \cup \{j\}$ must intersect S^*, and since these $k+1$ balls are disjoint, it must be that $|S^*| > k$. Therefore, c_{jS} is within a factor 2 of any solution with size at most k. Hsu & Nemhauser [15] showed that, for any $\epsilon > 0$, there is

no polynomial-time algorithm with a performance guarantee of $2 - \epsilon$ for this problem, unless P=NP.

Another example of a cost vector that is approximable within a constant factor would be that of the k-median problem. In this case, the cost vector w is 1 for each index, and therefore the cost is simply the sum of the distances between each demand point and its closest open service point. Although the k-median problem is arguably driven by the simplest cost vector, no constant approximation algorithms were known for it until Charikar *et al.* [8] presented an LP-based algorithm with a factor of $6\frac{2}{3}$. Indeed, our main result here extends their approach. Subsequent to [8], there has been a series of improvements leading to the recent $1 + \sqrt{3} + \epsilon$ approximation algorithm by Li and Svensson [22].

In this chapter, we push the envelope a little further by showing that even if the cost vector is a convex combination of the cost vectors of k-center and k-median, one can still efficiently find solutions within a constant factor of the optimal. More precisely, given an instance of k-supplier with an optimal solution S_{opt}, our algorithm presented in Section 5.2 is able to find a solution S such that $\text{sum}(c_{\mathcal{D}S}) \leq 8\text{sum}(c_{\mathcal{D}S_{\text{opt}}})$ and $\max(c_{\mathcal{D}S}) \leq 4\max(c_{\mathcal{D}S_{\text{opt}}})$. In the literature, this is referred to as a *bicriteria approximation algorithm*. Note that this result implies that for the special case of the ordered median problem in which w is obtained from a convex combination of the k-median and k-center problems, we obtain a constant approximation algorithm.

A stronger notion than a bicriteria approximation would be a simultaneous approximation; that is, an algorithm that produces one common solution S that is simultaneously compared to an optimal solution S_1^* for its k-center objective and to an optimal solution S_2^* for its k-median objective. However, we can show

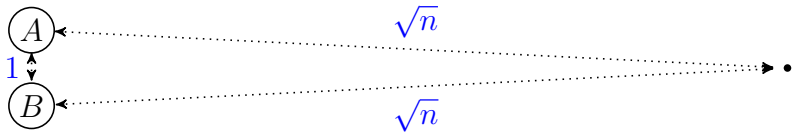

Figure 5.1: An example showing that simultaneous approximation of k-center and k-median within a factor $o(\sqrt{n})$ is impossible. Here each of A and B represents a cluster of $\lfloor \frac{n}{2} \rfloor$ points that are located at the same position.

that producing such a simultaneous approximation of factor $o(\sqrt{n})$ is not possible. For this, suppose n is odd and $n-1$ of the points are divided equally between two positions with distance 1 from each other. There is also a single outlier that is of distance $\sqrt{n}$ from everyone else. See Figure 5.1. Let $k = 2$ and suppose S_1^* contains the outlier and one other point, whereas S_2^* contains one point at each of the densely populated locations. Therefore, $\max(c_{\mathcal{D}S_1^*}) = 1$ and $\text{sum}(c_{\mathcal{D}S_2^*}) = \sqrt{n}$. In such a case, any solution S would have $\text{sum}(c_{\mathcal{D}S}) \geq \lfloor \frac{n}{2} \rfloor$ if it opens the outlier service point and $\max(c_{\mathcal{D}S}) \geq \sqrt{n}$ otherwise. In either case, $\max(\frac{\max(c_{\mathcal{D}S})}{\max(c_{\mathcal{D}S_1^*})}, \frac{\text{sum}(c_{\mathcal{D}S})}{\text{sum}(c_{\mathcal{D}S_2^*})})$ would be in $\Omega(\sqrt{n})$.

We also extend our result to the incremental setting, when k is not known *a priori* and one has to add service points one by one to the solution such that when the size of the solution reaches k, the solution that is built is still within a constant factor of the optimal. For example, the algorithm that we mentioned for the k-center problem satisfies this constraint, since it opens service points greedily and oblivious of k. For the k-median problem, Mettu and Plaxton [27] presented the first constant-factor approximation in the incremental setting. Later, Lin *et al.* [26] presented a general framework for incremental approximations that also applies to the k-median problem with an improved approximation factor. In Section 5.3, we use this framework to develop an incremental approximation algorithm for any objective that is a convex combination of that of k-center and k-median problems.

5.2 A (4,8)-approximation algorithm for k-center & k-median

Suppose we are given an instance of the k-supplier problem with $\mathcal{D} = \mathcal{F}$, and we wish to find a good solution with respect to both the k-median and k-center objectives. Let S_{opt} be the "ideal" solution in this context, and consider $\max(c_{\mathcal{D}S_{\text{opt}}})$ and $\text{sum}(c_{\mathcal{D}S_{\text{opt}}})$, which are its k-center and k-median objectives, respectively. As is the convention in bi-criteria approximations, we assume that $\max(c_{\mathcal{D}S_{\text{opt}}})$ is given as an objective measure. Note that $\max(c_{\mathcal{D}S_{\text{opt}}})$ is equal to one of the $\binom{n}{2}$ possible distances between pairs of given points, and hence we can guess $\max(c_{\mathcal{D}S_{\text{opt}}})$ with a multiplicative overhead of $O(n^2)$. This guess would allow us to achieve the desired guarantees, however, we would not necessarily be able to decide which guess corresponds to the true value of $\max(c_{\mathcal{D}S_{\text{opt}}})$ and would need to choose among produced solutions based some other criteria.

Our algorithm works based on the k-supplier LP. In this formulation, for a service point $i \in \mathcal{F}$ there is a variable y_i that represents whether service point i is opened or not, and a variable x_{ij} for each $j \in \mathcal{D}$ that represents the fraction of j's demand that is satisfied by i. Here we are assuming that each demand point has one unit of demand that must be satisfied. However, the algorithm will maintain a more general instance in which demand is concentrated at nodes in $\mathcal{D}$, and so we will let d_j denote the total demand at node j.

Suppose that variables x_{ij} and y_{ij} are restricted to be integers and $\mathcal{L}$ is set to infinity. Then the (k-supplier LP) formulation represents an linear relaxation of the k-median problem: the objective is to minimize the sum of distances between each demand point and the service point that serves it. Constraint (c1) ensures that a demand point $j \in \mathcal{D}$ relies on exactly one service point $i \in \mathcal{F}$,

constraint (c2) verifies that no demand point uses an unopened service point to satisfy its demand, and constraint (c3) makes sure that k centers are opened.

$$\min \quad \sum_{i,j \in \mathcal{D}} d_j c_{ij} x_{ij} \qquad \text{(k-supplier LP)}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{F}} x_{ij} = 1 \qquad \forall j \in \mathcal{D} \qquad \text{(c1)}$$

$$x_{ij} \leq y_i \qquad \forall i \in \mathcal{F}, j \in \mathcal{D} \qquad \text{(c2)}$$

$$\sum_{i \in \mathcal{F}} y_i = k \qquad \text{(c3)}$$

$$0 \leq y_i \leq 1 \qquad \forall i \in \mathcal{F}$$

$$0 \leq x_{ij} \leq 1 \qquad \forall i \in \mathcal{F}, j \in \mathcal{D}$$

$$x_{ij} = 0 \qquad \forall i \in \mathcal{F}, j \in \mathcal{D} : c_{ij} > \mathcal{L} \qquad \text{(cc)}$$

Now suppose we set $\mathcal{L}$ to $\max(c_{\mathcal{D}S_{\text{opt}}})$ and then solve for an optimal fractional solution (x^*, y^*). Then constraint (cc) guarantees that x_{ij} is 0 for any pair $i \in \mathcal{F}$ and $j \in \mathcal{D}$ that are further than $\max(c_{\mathcal{D}S_{\text{opt}}})$ from each other. Also, by the optimality of (x^*, y^*) and since S_{opt} corresponds to a feasible solution for (k-supplier LP), we have that $\sum_{i,j \in \mathcal{D}} d_j c_{ij} x^*_{ij} \leq \text{sum}(c_{\mathcal{D}S_{\text{opt}}})$. Thus, we can use (x^*, y^*) to bound both the k-center and k-median objectives in the optimal solution.

Algorithm 3 Finds an integral solution S with $\mathrm{sum}(c_{\mathcal{D}S})$ at most 8 times (k-supplier LP) objective and $\max(c_{\mathcal{D}S})$ at most $4\mathcal{L}$.

INPUT: An instance of the k-supplier problem along with $\mathcal{L}$ and a feasible solution (x, y) of the (k-supplier LP).

OUTPUT: A set S with $|S| \leq k$ of service points that will be opened.

$(x', y') \leftarrow (x, y)$
$S' \leftarrow \emptyset$
$d'_j \leftarrow d_j \quad \forall j \in \mathcal{D}$ {total demand that j represents}
$\overline{C}_j \leftarrow \sum_{i \in \mathcal{F}} x_{ij} c_{ij} \quad \forall j \in \mathcal{D}$ {cost of serving the demand of j}
for all demand points $j \in \mathcal{D}$ in increasing order of $\overline{C}_j$ **do**
  if $c_{jS'} > \min(4\overline{C}_j, 2\mathcal{L})$ **then**
    $S' \leftarrow S' \cup \{j\}$
  else
    let j' be the closest point to j in S'
    $d'_{j'} \leftarrow d'_{j'} + d'_j$, $d'_j \leftarrow 0$ {moving the demand of j to j'}
  end if
end for
for all $i \in \mathcal{F} \backslash S'$ **do**
  let i' be the closest point to i in S'
  $y'_{i'} \leftarrow y'_{i'} + y'_i$, $y'_i \leftarrow 0$ {moving the y' value of j to j'}
end for
while there is $i, i' \in S'$ s.t. $y'_{i'} < 1$, $y'_i > 1$ **do**
  $\delta \leftarrow \min(y'_i - 1, 1 - y'_{i'})$
  $y'_{i'} \leftarrow y'_{i'} + \delta$, $y'_i \leftarrow y'_i - \delta$ {redistributing the excess y' of i}
end while
let s_j be the closest point to j in $S' \backslash \{j\} \quad \forall j \in S'$
while there are $i, i' \in S'$ s.t. $y'_{i'} < 1$, $\frac{1}{2} < y'_i < 1$, and $d'_i c_{is_i} \leq d'_{i'} c_{i's_{i'}}$ **do**
  $\delta \leftarrow \min(y'_i - \frac{1}{2}, 1 - y'_{i'})$
  $y'_{i'} \leftarrow y'_{i'} + \delta$, $y'_i \leftarrow y'_i - \delta$
end while
$S \leftarrow \{i \in S' : y'_i = 1\}$
let T be a forest of arbitrarily rooted trees with a node for each $j \in S'$ and an edge between each $j \in S'$ and s_j
let E and O be the set of points $j \in S'$ with $y'_j < 1$ that are of even and odd distance to their tree root in T, respectively
if $|O| < |E|$ **then**
  $S \leftarrow S \cup O$
else
  $S \leftarrow S \cup E$
end if

return S

Algorithm 3 uses a solution (x^*, y^*) to construct an integral solution that is within a constant factor of both the k-center and k-median objectives of S_{opt}. First, a set S' of points is selected as candidates for being opened, and at the same time, the demand of each point $j \in \mathcal{D} \backslash S'$ is moved to a point $j' \in S'$ that is close enough to represent j (lines 5-12). Here, for each $j \in S'$,d'_j denotes the total demand that point j represents. Then, another loop iterates over all points $i \in \mathcal{F} \backslash S'$ to create an updated solution y'; we move all of the weight y'_i to the closest point in S' so that $y'_i = 0$ for each $i \in \mathcal{F} \backslash S'$, and then a while loop redistributes among S' any excess of y' that might have been gathered in one $i' \in S'$ so that $y'_{i'} \leq 1$ (lines 13-20). By this point in the algorithm, all of the demand and (fractional) supply is concentrated in S'. The following lemma implies that the size of S' is at most $2k$.

Lemma 5. *After Line 20 of Algorithm 3 we have $y'_i > \frac{1}{2}$ for each $i \in S'$.*

Proof. By the order in which the points are considered in the loop at line 5 and the condition at line 6, we have that the balls centered at any $i \in S'$ with radius $\min(2\overline{C}_i, \mathcal{L})$ are disjoint. (Recall that $\overline{C}_j = \sum_{i \in \mathcal{F}} x_{ij} c_{ij}$ given our input fractional solution x.) By line 20 of the algorithm, all of the y value that is in any such a ball centered at an $i \in S'$ is moved to y'_i, and some of it is possibly redistributed to make sure $y'_i \leq 1$. If $\mathcal{L} < 2\overline{C}_i$ and the radius of the ball is $\mathcal{L}$, then by constraint (cc), the total y value in such a ball must be at least 1. Otherwise, by constraint (c2) and the definition of $\overline{C}_i$, at least half of the y value that i uses to satisfy its demand must be within distance $2\overline{C}_i$. □

Next, in a while loop, the algorithm moves any weight specified by y' from any point of S' to a more effective point in S' as long as all y' values remain in range $[\frac{1}{2}, 1]$ for all points in S' (lines 22–25).

Lemma 6. *By the time the while loop at line 22 of Algorithm 3 terminates, for any point $i \in S'$, we have $y'_i \in \{\frac{1}{2}, 1\}$.*

Proof. Since constraint (c3) holds with equality and (x^*, y^*) is a feasible solution to the (k-supplier LP), we have $\sum_{i \in \mathcal{F}} y^*_i = k$. Since y' is initialized to y^* and the total value of y' is conserved throughout (both when it is moved to S' and also in each iteration of the while loop,) therefore, any time the condition of the while loop at line 22 is being evaluated, we have

$$\sum_{i \in \mathcal{F}} y'_i = \sum_{i \in S'} y'_i = k.$$

So if the condition of the lemma is not satisfied, then there must be at least two points $i, i' \in S'$ such that $y'_i, y'_{i'} \in (\frac{1}{2}, 1)$. However, such a pair satisfies the condition of the while loop and therefore the loop would not yet terminate. □

After the while loop at line 22 terminates, all points $i \in S'$ with $y'_i = \frac{1}{2}$ are partitioned into two sets O and E, and the smaller of O and E along with all the points $i \in S'$ with $y'_i = 1$ are returned as the set S of points that are to be opened by the algorithm (lines 27–35). Since S is comprised of points $i \in S'$ with $y'_i = 1$ and at most half of the points $i' \in O \cup E$ with $y'_{i'} = \frac{1}{2}$, therefore the size of S is no larger than k. The following lemmas argue that this set S is indeed a bicriteria approximate solution for k-center and k-median.

Lemma 7. *Given an instance of k-supplier problem with $\mathcal{D} = \mathcal{F}$ and a corresponding optimal solution (x^*, y^*) of (k-supplier LP) with $\mathcal{L} = \max(c_{\mathcal{D}S_{\text{opt}}})$, Algorithm 3 returns a set S of at most k centers such that $\max(c_{\mathcal{D}S}) \leq 4\max(c_{\mathcal{D}S_{\text{opt}}})$.*

Proof. First we argue that for each $j \in \mathcal{D}$, the closest point to j that is added to S' is at most of distance $2\mathcal{L}$ from j. For this, note that when the for loop at line 5

of the algorithm iterates over j and the condition at line 6 is being evaluated, if the distance from j to all points in S' is greater than $2\mathcal{L}$, then j itself would have been added to S'.

Suppose that j' is the closest node in S' to j. Let j'' be the closest node in S' to j', other than j' itself. If the distance between j' and j'' is greater than $2\mathcal{L}$, then all of the y' value in the ball of radius $\mathcal{L}$ around j' must have been gathered at j' in lines 13–16, since j' is the closest point in S' to any point in this ball. Since the solution (x^*, y^*) is feasible and by constraint (cc), the total amount of y' value in such a ball should at least be 1 in order to satisfy the demand of j'. Therefore, in this case $y'_{j'}$ ends up being 1 and hence j' itself is opened by the algorithm.

Now suppose that the distance of j' and j'' is smaller than $2\mathcal{L}$. In this case, we show that at least one of j' and j'' is opened by the algorithm. By the triangle inequality, this would mean that there must be at least one opened service point within distance $4\mathcal{L}$ of j, concluding the proof of the lemma. By the construction of the forest T in line 27, there is an edge between nodes representing j' and j'' in T. If none of j' and j'' are added to S at line 26, then they are either in O or E. Since there is an edge between j' and j'' in T, the parity of their level in T must be different, and hence exactly one of them should belong to O and the other must be in E. One of O or E is added to S in lines 29–35, and hence one of j' or j'' must be opened by the algorithm. □

Lemma 8. *Given an instance of k-supplier problem with $\mathcal{D} = \mathcal{F}$ and a corresponding optimal solution (x^*, y^*) of (k-supplier LP) with $\mathcal{L} = \max(c_{\mathcal{D}S_{\text{opt}}})$, Algorithm 3 returns a set S of at most k centers such that $\text{sum}(c_{\mathcal{D}S}) \leq 8\text{sum}(c_{\mathcal{D}S_{\text{opt}}})$.*

Proof. To show this we can break down the cost of rounding the LP solution into 3 parts. The first part (filtering phase at Lines 5–12) incurs an independent

additive factor of 4, while the second (Lines 13–20) and third (Line 27 onward) parts each incur a multiplicative factors of 2, implying that $\mathrm{sum}(c_{\mathcal{D}S}) \leq (4 + 2 \times 2)\mathrm{sum}(c_{\mathcal{D}S_{\mathrm{opt}}})$.

We claim that moving the demands in Lines 5–12 induces an additive term of 4 to the approximation factor. To see this recall that $\mathrm{sum}(c_{\mathcal{D}S_{\mathrm{opt}}}) \geq \sum_{j\in\mathcal{D}} \sum_{i\in\mathcal{F}} x^*_{ij} c_{ij} = \sum_{j\in\mathcal{D}} \overline{C}_j$, and notice that the demand of each point j is moved a distance of at most $4\overline{C}_j$ and by the triangle inequality, returning this demand to its original position would incur at most a cost of $4\overline{C}_j$. Hence, the total overhead of moving the demands is within a factor 4 of the cost of the LP relaxation.

Moving the y' values to nodes at S' at Lines 13–20 incurs a factor of 2 to the approximation ratio. Suppose an ϵ amount of y is moved from i to $i' \in S'$ at Line 15. Note that at this point of the algorithm only points in S' have positive demand, and consider some $j \in S'$ that used to rely on i for an ϵ fraction of its demand, who now has to use i' to satisfy that ϵ fraction of its demand. By the choice of i' we have $c_{ii'} \leq c_{ji}$; therefore, by triangle inequality we have $c_{ji'} \leq c_{ji} + c_{ii'} \leq 2c_{ij}$.

The while loop at Line 22 chooses i and i' and manipulates the y' values such that the objective $\sum d'_j x'_{ij} c_{ij}$ does not increase; thus the bound on the approximation ratio is unchanged.

Suppose we update x' at this point in the algorithm (after termination of the while loop at Line 22) to get a feasible solution to the (k-supplier LP) while minimizing the objective $\sum d'_j x'_{ij} c_{ij}$ according to the new y' values produced by the algorithm. Then, by Lemma 6, at line 27 of the algorithm, each point j in S'

with $y'_j < 1$ has $y'_j = \frac{1}{2}$, and therefore relies for exactly half of its demand on the closest other point in S'; that is, s_j as defined in Line 21. By the same arguments as in the proof of Lemma 7, at the end of the algorithm if j is not opened, then s_j must be opened, and since j already relies on s_j for half of its demand, this would at most double the cost of serving point j. With the additive factor of 4 that we reserved for moving back the demands to their original points, it all adds up to a factor of 8. □

The following theorem immediately follows from Lemmas 7 and 8:

Theorem 9. *There is a polynomial-time bicriteria $(4, 8)$-approximation algorithm for k-center and k-median problems.*

5.3 Incremental approximation for convex combination of k-center and k-median

In the incremental setting, the parameter k is not known *a priori*. In this case, we open service points one by one, and the true value of k is revealed only after the kth service point is opened. One can view this setting equivalently as producing an ordering of all nodes in $\mathcal{F}$, where each prefix of the first k nodes is the designated solution corresponding to the number k, and all such prefixes for all $1 \leq k \leq n$ must have the claimed performance guarantee. For this case, we use the framework developed by Lin, Nagarajan, Rajaraman, and Williamson [26] to derive our result. The following is a corollary of their main theorem as it pertains to our problem.

Corollary 10. *[26] If there is an efficient procedure that, given a set of centers S*

and a number $k' < |S|$, finds a set $S' \subset S$ with $|S'| \leq k'$ such that $\max(c_{\mathcal{D}S'}) < \max(c_{\mathcal{D}S}) + \alpha_1 \max(c_{\mathcal{D}S_{\text{opt}}})$ and $\text{sum}(c_{\mathcal{D}S'}) < \text{sum}(c_{\mathcal{D}S}) + \alpha_2 \text{sum}(c_{\mathcal{D}S_{\text{opt}}})$ for any S_{opt} with $|S_{\text{opt}}| \leq k'$, then there is a randomized $(e \max(\alpha_1, \alpha_2))$-approximation algorithm and a deterministic $(4 \max(\alpha_1, \alpha_2))$-approximation algorithm for the incremental version of the problem where the objective is a convex combination of the objectives for the k-center and k-median problems.

To get a sense of how this algorithm works, suppose we wish to construct a chain $S_0 \subset S_1 \subset S_2 \subset S_3 \subset \ldots \subset S_l$ with $S_0 = \emptyset$ and $S_l = \mathcal{F}$, such that an incremental algorithm would only have to open the points in $S_i \backslash S_{i-1}$ before $S_{i+1} \backslash S_i$ for any $i \in \{1, 2, \ldots, l-1\}$ to be within the constant factor of the optimal solution for any intermediate value of k. To find this chain, one can start with S_l which is simply the set of all given points, and recursively use the procedure described in Corollary 10 to find a subset of the current set that is of cost roughly twice the current set. The following lemma describes such a procedure and proves its properties.

Lemma 11. *There is an efficient procedure that given a set of centers S and a number $k' < |S|$ finds a set $S' \subset S$ with $|S'| \leq k'$ such that $\max(c_{\mathcal{D}S'}) \leq \max(c_{\mathcal{D}S}) + 4\max(c_{\mathcal{D}S_{\text{opt}}})$ and $\text{sum}(c_{\mathcal{D}S'}) \leq \text{sum}(c_{\mathcal{D}S}) + 16\text{sum}(c_{\mathcal{D}S_{\text{opt}}})$ for any S_{opt} with $|S_{\text{opt}}| \leq k'$.*

Proof. To find S', we first use Theorem 9 with k set to k' and construct a solution $\hat{S}$ with $\max(c_{\mathcal{D}\hat{S}}) \leq 4\max(c_{\mathcal{D}S_{\text{opt}}})$ and $\text{sum}(c_{\mathcal{D}\hat{S}}) \leq 8\text{sum}(c_{\mathcal{D}S_{\text{opt}}})$. For each point j, let $n(j)$ and $\hat{n}(j)$ be the closest point to j in S and $\hat{S}$, respectively. Then we define S' simply as the union of all points in S that are closest to some point in $\hat{S}$, that is, $S' = \{n(j) : j \in \hat{S}\}$. Next we show that S' indeed satisfies the properties claimed in the lemma.

First note that since $n(\hat{n}(j)) \in S'$ for each point j, we have that $c_{jS'} \leq c_{jn(\hat{n}_j)}$, and, by the triangle inequality, we have that $c_{jn(\hat{n}_j)} \leq c_{j\hat{n}(j)} + c_{\hat{n}(j)n(\hat{n}_j)}$. Since $c_{j\hat{n}(j)} \leq \max(c_{\mathcal{D}\hat{S}})$ and $c_{\hat{n}(j)n(\hat{n}_j)} \leq \max(c_{\mathcal{D}S})$, therefore we have that

$$\max(c_{\mathcal{D}S'}) \leq \max(c_{\mathcal{D}S}) + \max(c_{\mathcal{D}\hat{S}}) \leq \max(c_{\mathcal{D}S}) + 4\max(c_{\mathcal{D}S_{\text{opt}}}).$$

For the other side of the argument, note that since $n(\hat{n}_j)$ is by definition the closest point in S to $\hat{n}(j)$, we have that $c_{\hat{n}(j)n(\hat{n}_j)} \leq c_{\hat{n}(j)n(j)}$. Therefore, by the triangle inequality, we have that

$$c_{jn(\hat{n}(j))} \leq c_{j\hat{n}(j)} + c_{\hat{n}(j)n(\hat{n}(j))} \leq c_{j\hat{n}(j)} + c_{\hat{n}(j)n(j)} \leq c_{j\hat{n}(j)} + c_{j\hat{n}(j)} + c_{jn(j)}.$$

Putting these together, we get that $c_{jS'} \leq c_{jn(\hat{n}(j))} \leq c_{jn(j)} + 2c_{j\hat{n}(j)}$ for each $j \in \mathcal{D}$. Thus,

$$\text{sum}(c_{\mathcal{D}S'}) \leq \text{sum}(c_{\mathcal{D}S}) + 2\text{sum}(c_{\mathcal{D}\hat{S}}) \leq \text{sum}(c_{\mathcal{D}S}) + 16\text{sum}(c_{\mathcal{D}S_{\text{opt}}}).$$

□

The following theorem immediately follows from Corollary 10 and Lemma 11:

Theorem 12. *The incremental approximation of the problem of minimizing a convex combination of the objectives of the k-center and k-median problems admits a randomized $16e$-approximation algorithm and a deterministic 64-approximation algorithm.*

The k-center and k-median problems are, in many respects, antipodal extremes among possible weighting functions for the order median problem; since we have shown that any convex combination of those two weighting functions can be approximated within a constant factor of optimal, we believe that it is natural to conjecture that such a result can be obtained for the general ordered median problem as well.

BIBLIOGRAPHY

[1] An, H.C., Singh, M., Svensson, O.: LP-based algorithms for capacitated facility location. In: Proc., 55th FOCS. pp. 256–265. (2014)

[2] Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM Journal on Computing 33(3), 544–562 (2004)

[3] Aouad, A., Segev, D.: The ordered k-median problem: Surrogate models and approximation algorithms. In submission.

[4] Bansal, M., Garg, N., Gupta, N.: A 5-approximation for capacitated facility location. In: Proc., 20th ESA. pp. 133–144. (2012)

[5] Barman, S., Chawla, S.: Traffic-redundancy aware network design. In: Proc., 23rd SODA. pp. 1487–198 (2011)

[6] Bienkowski, M., Byrka, J., Chrobak, M., Dobbs, N., Nowicki, T., Sviridenko, M., wirszcz, G., Young, N.: Approximation algorithms for the joint replenishment problem with deadlines. Journal of Scheduling 18(6), 545–560 (2015)

[7] Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location problems. SIAM J. Computing 34(4), 803–824 (2005)

[8] Charikar, M., Guha, S., Tardos, É., Shmoys, D.B.: A constant-factor approximation algorithm for the k-median problem. J. Comput. Syst. Sci., 65(1):129–149 (2002)

[9] Cheung, M., Elmachtoub, A.N., Levi, R., Shmoys, D.B.: The submodular joint replenishment problem. Math. Prog. (2014)

[10] Chudak, F.A., Shmoys, D.B.: Improved approximation algorithms for the uncapacitated facility location problem. SIAM Journal on Computing 33(1), 1–25 (2003)

[11] Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. Mathematical programming 102(2), 207–222 (2005)

[12] Gonzalez, T.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38:293–306, (1985)

[13] Grötschel, M., Lovász, L., Schrijver, A.: The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1(2), 169–197 (1981)

[14] Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Math. Oper. Res., 10(2):180–184 (1985)

[15] Hsu, W., Nemhauser G.L.: Easy and hard bottleneck location problems. Discrete Applied Mathematics, 1(3):209–215, (1979)

[16] Jain, K., Mahdian, M., Markakis, E., Saberi, A., Vazirani, V.V.: Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. JACM 50(6), 795–824 (2003)

[17] Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. JACM 48(2), 274–296 (2001)

[18] Korupolu, M.R., Plaxton, C.G., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. Journal of algorithms 37(1), 146–188 (2000)

[19] Levi, R., Lodi, A., Sviridenko, M.: Approximation algorithms for the capacitated multi-item lot-sizing problem via flow-cover inequalities. Mathematics of Operations Research 33(2), 461–474 (2008)

[20] Levi, R., Roundy, R., Shmoys, D., Sviridenko, M.: A constant approximation algorithm for the one-warehouse multiretailer problem. Management Science 54(4), 763–776 (2008)

[21] Levi, R., Roundy, R.O., Shmoys, D.B.: Primal-dual algorithms for deterministic inventory problems. Math. of OR. 31(2), 267–284 (2006)

[22] Li, S., Svensson, O.: Approximating k-median via pseudo-approximation. SIAM J. Comput., 45(2):530–547 (2016)

[23] Li, S. : Constant Approximation Algorithm for Non-Uniform Capacitated Multi-Item Lot-Sizing via Strong Covering Inequalities. In: Proc., 28th SODA. pp. 2311–2325. (2017).

[24] Li, S.: A 1.488 approximation algorithm for the uncapacitated facility location problem. Information and Computation 222, 45–58 (2013)

[25] Li, S., Svensson, O.: Approximating k-median via pseudo-approximation. In: Proc., 45th STOC. pp. 901–910. (2013)

[26] Lin, G., Nagarajan, C., Rajaraman, R., Williamson, D.P.: A general approach for incremental approximation and hierarchical clustering. SIAM J. Comput., 39(8):3633–3669, Nov. (2010)

[27] Mettu, R.R., Plaxton, C.G.: The online median problem. SIAM J. Comput., 32(3):816–832 (2003)

[28] Pal, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: Proc. 42nd FOCS. pp. 329–338. (2001)

[29] Schrijver, A.: Combinatorial optimization: polyhedra and efficiency, vol. 24. Springer Science & Business Media (2003)

[30] Shmoys, D.B., Tardos, É., Aardal, K.: Approximation algorithms for facility location problems. In: Proc. 29th STOC. pp. 265–274. (1997)

[31] Swamy, C.: Improved approximation algorithms for matroid and knapsack median problems and applications. In Proc. 17th APPROX, pp. 403–418. (2013)

[32] Williamson, D.P., Shmoys, D.B.: The design of approximation algorithms. Cambridge University Press (2011)