ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Approximation Algorithms for Online Facility Location and Radii Facility Location

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευαγγελία-Σοφία Γεργατσούλη

Επιβλέπων: Δημήτρης Φωτάκης
Επίκουρος Καθηγητής ΕΜΠ

Αθήνα, Ιούνιος 2018

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Approximation Algorithms for Online Facility Location and Radii Facility Location

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευαγγελία-Σοφία Γεργατσούλη

Επιβλέπων: Δημήτρης Φωτάκης
Επικουρος Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή Ιούνιος 2018

.....	.....	.....
Δημήτρης Φωτάκης	Νικόλαος Παπασπύρου	Αριστείδης Παγουρτζής
Επίκουρος Καθηγητής ΕΜΠ	Αναπληρωτής Καθηγητής ΕΜΠ	Αναπληρωτής Καθηγητής ΕΜΠ

.....
Ευαγγελία-Σοφία Γεργατσούλη
(Διπλωματούχος Ηλεκτρολόγος Μηχανικός & Μηχανικός Υπολογιστών Ε.Μ.Π.)

Οι απόψεις που εκφράζονται σε αυτό το κείμενο είναι αποκλειστικά του συγγραφέα και δεν αντιπροσωπεύουν απαραίτητα την επίσημη θέση του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Τα προβλήματα χωροθέτησης (Facility Location) είναι ένα από τα κλασσικά προβλήματα στη συνδυαστική βελτιστοποίηση, το οποίο έχει μελετηθεί από πολλές διαφορετικές σκοπιές και με πολλούς διαφορετικους περιορισμούς και παραλλαγές. Σύντομα, όσοι μελετούσαν το πρόβλημα, συνειδητοποίησαν τη δυσκολία του, έτσι αντί να προσπαθούν να βρουν ακριβή λύση, το οποίο θα ήταν πολύ χρονοβόρο για να έχει πρακτική εφαρμογή, προσπαθούσαν να προσεγγίσουν την βέλτιστη λύση σε πολυωνυμικό χρόνο. Αλλα παρόλο που μπορούμε να βρούμε σταθερούς προσεγγιστικούς αλγορίθμους, δεν είναι πολύ ρεαλιστικό να υποθέτουμε ότι γνωρίζουμε όλα τα δεδομένα εισόδου εξ αρχής. Σκεφτείτε ένα δίκτυο υπολογιστών για παράδειγμα, όπου νέοι κόμβοι προστίθενται ή αφαιρούνται συνεχώς. Αυτή η "αβεβαιότητα" στα δεδομένα εισόδου μας οδήγησε στο να ορίσουμε το πεδίο των *online αλγορίθμων*, και το πρόβλημα Facility Location είναι ένα από αυτά που μελετήθηκαν σε αυτό το "αβεβαιο" περιβάλλον. Με αυτήν την υπόθεση αβεβαιότητας, όπου ο αλγόριθμος δεν γνωρίζει εξ αρχής όλα τα δεδομένα που θα κληθεί να επεξεργαστεί, είναι αδύνατον να λύσει το πρόβλημα με το βέλτιστο τρόπο, πόσο μάλλον για ένα πρόβλημα που είναι έτσι κι αλλιώς υπολογιστικά δύσκολο να λυθει βέλτιστα. Χρησιμοποιώντας competitive ανάλυση, μπορούμε να μετρήσουμε την απόδοση ενος τέτοιου αλγορίθμου και να δώσουμε εγγυήσεις ότι η λύση μας δε θα είναι πολύ μακριά από τη βέλτιστη. Στην παρούσα διπλωματική παρουσιάζονται οι βασικές έννοιες γενικά για online και προσεγγιστικους αλγορίθμους και ενα σημαντικό κομμάτι της δουλειάς που έχει ήδη γίνει σε online facility location και παραλλαγές του. Τέλος, παρουσιάζεται μια νέα παραλλαγή του online προβλήματος με έναν νεο αλγόριθμο για αυτήν.

Λέξεις κλειδιά: Facility Location, Sum Radii, προσεγγιστικοί αλγόριθμοι, online αλγόριθμοι, clustering, competitive analysis

Abstract

Facility Location is a classic problem in combinatorial optimization, and has been studied many years, in many different contexts and with various additions and modifications. Soon, computer scientists realized the hardness of the problem, so instead of trying to find an exact solution, which would be too time consuming to be useful in solving real life problems, we can approximate the optimal exact solution in polynomial time. But, even though we can find constant approximation algorithms for this problem, it is not very realistic to assume that we know all the input data from the beginning. Consider a computer network for example, where new nodes are constantly added, or deleted. This "uncertainty" in the input data has led us to create the field of *online algorithms*, and Facility Location is one of the problems to be studied under uncertainty. In this setting, where the algorithm does not know all the data in advance, it cannot possibly make the best decision, so much so for a problem that even with all the data available is computationally hard. Using competitive analysis, we measure the performance of such an algorithm, and provide guarantees that it will not be far from the optimal. In this thesis we present the basic notions about approximation and online algorithms and parts of the previous work for the Online Facility Location problem, and some interesting variants. Finally, we will present a new variant called *Radii Facility Location*, provide an algorithm for this, and discuss how the competitive ratio changes for some parameters of the problem.

Keywords: Facility Location, Sum Radii, approximation algorithms, online algorithms, competitive analysis, clustering

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον κ. Φωτάκη, γιατί ήταν εκείνος που με έκανε να θέλω να ασχοληθώ με τη θεωρητική πληροφορική μέσα από το μάθημα του και τον ενθουσιασμό του για αυτό που κάνει. Ελπίζω να συνεχίσει να εμπνέει κόσμο να ασχοληθεί με τους αλγορίθμους, όπως έκανε και με εμένα. Τον κ. Παπασπύρου γιατί, παρόλο που δεν ακολούθησα τα "λιγότερο θεωρητικά" της πληροφορικής όπως αρχικά σκεφτόμουν, τα μαθήματα και κυρίως οι διαλέξεις του ηταν από τα καλύτερα πράγματα στη σχολή. Τον Λουκά Κ. για τις μακροσκελεις και ακρως εποικοδομητικές μας συζητησεις πάνω στο πρόβλημα της παρούσας διπλωματικής. Επίσης τα μέλη του corelab, και τους κ.Ζαχο και κ.Παγουρτζη για την καθοδήγηση και τις συμβουλές τους.

Θα ήθελα να ευχαριστήσω ακόμα τα παιδιά από το καλύτερο εργαστήριο της σχολής (γνωστό και ως ΜΟΠ) για την καταπληκτική ατμόσφαιρα, και τις ομάδες της βιβλιοθήκης ΗΜΜΥ και του shmmy.ntua.gr για τις ωραίες στιγμές που περάσαμε και όσα κάναμε μαζί, συνεχίστε την καλή δουλειά παιδιά!

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου και τους φίλους Γιωργο Κ, Αννα Μ, Ειρηνη Κ, Αννα Γ, Νανσυ Α, Αννα Ο, Μαρια Χ, Ηλιανα Λ, Νίκο Ζ, Αλέξανδρο Τ, Βασίλη Κ, Ελένη Ψ για την υπέροχη παρέα τους όλα αυτά τα χρόνια.

Ε.Γ.

"Τα quotes στις διπλωματικές δεν ειναι cringy."

— Α. Τσιγώνιας

Contents

Chapter 1

Εκτεταμένη Ελληνική Περίληψη

Δίνουμε μια εκτεταμένη ελληνική περίληψη που συνοψίζει το περιεχόμενο αυτής της διπλωματικής. Θα παρουσιαστούν συνοπτικά τα περιεχόμενα κάθε κεφαλαίου, χωρις αποδείξεις και τεχνικές λεπτομέρειες.

1.1 Τεχνικό Υπόβαθρο

Σε αυτό το κομμάτι παρουσιάζονται κάποια βασικά τεχνικά εργαλεία που χρησιμοποιούνται αργότερα στη διπλωματική και υπάρχουν αναλυτικά στο κεφάλαιο 3. Αυτά είναι ο γραμμικός προγραμματισμός (linear programming), βασικές έννοιες και τεχνικές ανάλυσης και σχεδίασης προσεγγιστικων αλγορίθμων μέσα από το πρόβλημα χωροθέτησης εγκαταστάσεων (Facility Location) και κάποια βασικά πράγματα για Online αλγόριθμους.

1.1.1 Γραμμικός Προγραμματισμός

Ένα γραμμικό πρόγραμμα είναι το πρόβλημα της βελτιστοποίησης μιας γραμμικής συνάρτησης ικανοποιώντας ταυτόχρονα περιορισμούς ισότητων ή ανισοτήτων. Η κανονική μορφή ενός γραμμικού προγράμματος (LP) φαίνεται στον πίνακα 1.1.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i & \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq b_j & \forall j \in [m] \\ & x_i \geq 0 & \forall i \in [n] \end{aligned}$$

Table 1.1: Κανονική μορφή ενός πρωτεύοντος (P)

Αυτό το πρόγραμμα λέγεται πρωτεύον και έρχεται σε ζευγάρι με ένα δυϊκό (πίνακας 1.2). Οι βασικοί αλγόριθμοι επίλυσης LPs είναι ο Simplex, ο Ellipsoid και κάποιες Interior point μεθοδοι.

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} b_j y_j & \\ \text{subject to} \quad & \sum_{j=1}^{m} a_{ij} y_j \leq c_i & \forall i \in [n] \\ & y_j \geq 0 & \forall j \in [m] \end{aligned}$$

Table 1.2: Κανονική μορφή του Δυϊκού (D)

Το πρωτεύον και το δυϊκό ικανοποιούν κάποιες συνθήκες complementary slackness, καθώς και την ισχυρη δυϊκότητα, που λέει ότι η βέλτιστη λύση του πρωτέυντος και του δυϊκού ταυτίζονται. Αυτές οι σχέσεις θα χρησιμοποιηθούν στη μέθοδο πρωτεύοντος-δυϊκού για να βρεθεί λύση στο γραμμικό πρόγραμμα και να πάρουμε προσεγγιστικούς αλγορίθμους χωρίς ουσιαστικά να λυσουμε το LP.

Είναι σημαντικό να αναφέρουμε ότι συνήθως τα προβήματα μοντελοποιούνται σαν *ακέραια* προγράμματα και όχι σαν γραμμικά, δηλαδή οι μεταβλητές τους είναι ακέραιες ενώ σε ένα LP η λύση μπορεί να δώσει και μη ακέραιες λύσεις. Συνηθίζουμε όμως να "χαλαρώνουμε" (relax) το ακέραιο πρόγραμμα και να επιτρέπουμε μη ακέραιες τιμές στις μεταβλητές καθως τα LPs λύνονται σε πολυωνυμικό χρόνο ενώ τα ακέραια προγράμματα όχι.

1.1.2 Προσεγγιστικοί Αλγόριθμοι και Facility Location

Οι προσεγγιστικοί αλγόριθμοι προέκυψαν σαν αποτέλεσμα του διάσημου ερωτήματος P vs NP. Κάθως υπάρχουν προβλήματα που δεν μπορούμε να λύσουμε σε λιγότερο απο εκθετικό

χρόνο, μια λύση είναι να προσπαθούμε να τα προσεγγίσουμε. Ο στόχος του κλάδου των προσεγγιστικών αλγορίθμων είναι να δώσει μαθηματική απόδειξη για τον αλγοριθμο ότι θα μπορεί να βρει μια καλή προσέγγιστη της βέλτιστης λύσης.

Για αυτόν τον λόγο ορίζουμε εναν c-προσεγγιστικό αλγόριθμο ως έναν πολυωνυμικό αλγόριθμο που για κάθε είσοδο του προβλήματος βρίσκει μια λύση με τιμή το πολύ c φορές την βέλτιστη τιμή. Παρακάτω παρουσιάζεται το πρόβλημα Facility Location γραμμένο σε μορφή LP. Σε αυτό το πρόβλημα, μας δίνονται ένα σύνολο από facilities και ένα σύνολο από πελάτες τους οποίους θέλουμε να συνδέσουμε με τα facilities. Ο κάθε πελάτης για να συνδεθεί πληρώνει την απόσταση του, ενώ ένα facility για να ανοίξει πληρώνει ένα κόστος f. Στόχος μας είναι να ελαχιστοποιήσουμε το άθροισμα των κοστών ανοίγματος και των κοστων σύνδεσης.

minimize	$\sum_i y_i f_i + \sum_{i,j} x_{ij} c_{ij}$		maximize	$\sum_j a_j$	
subject to	$\sum_i x_{ij} \geq 1$	$, \forall j$	subject to	$a_j - b_{ij} \leq c_{ij}$	$, \forall j, i$
	$x_{ij} \leq y_i$	$, \forall i, j$		$\sum_j b_{ij} \leq f_i$	$, \forall i$
	$x_{ij} \geq 0$			$a_j \geq 0$	
	$y_i \geq 0$			$b_{ij} \geq 0$	

Table 1.3: To Facility Location σαν γραμμικό πρόγραμμα, $i \in F$ and $j \in C$ for the above

Οι μεταβλητές είναι ουσιαστικά δεικτες για το αν θα ανοίξει ένα facility i ($y_i = 1$ ανν το facility i ανοίγει), και για το αν ο πελάτης j είναι συνδεδεμένος με το facility i ($x_{ij} = 1$).

Οι βασικότερες τεχνικές επίλυσης τέτοιων προβλημάτων είναι η ντετερμινιστική και τυχαιοποιημένη στρογγυλοποίηση ενος LP, και η μέθοδος πρωτεύοντος-δυϊκού. Στην περίπτωση της στρογγυλοποίησης λύνουμε το LP και μετά προσπαθούμε να στρογγυλοποιήσουμε τις μεταβλητές σε ακέραιες τιμές, για να έχουμε μια λύση που να έχει νόημα για το πρόβλημα. Η διαφορά των δύο τεχνικών είναι στο αν η στρογγυλοποίηση γίνεται με κάποιον προκαθορισμένο τρόπο ή περιλαμβάνει κάποια τυχαία μεταβλητη. Στην περίπτωση της μεθόδου πρωτεύοντος-δυϊκού, προσπαθούμε να "χτισουμε" μια λύση στο δυϊκό ή στο πρωτεύον, ικανοποιώντας τις συνήκες complementary slackness. Ο καλύτερος λόγος προσέγγισης αυτη τη στιγμή είναι το 1.488 [1] με lower bound το 1.463 [2].

1.1.3 Online Αλγόριθμοι

Ενώ η παραδοσιακή προσέγγιση στους αλγορίμους λέει ότι γνωρίζουμε τα δεδομένα εισόδου απο την αρχή, αυτό συνήθως δε συμβαίνει σε πραγματικά προβλήματα. Για παράδειγμα σε ένα

δίκτυο μπορεί νέοι κόμβοι έρχονται συνέχεια και να πρέπει να εξυπηρετηθούν αμέσως, όπως στο πρόβλημα k-server. Ένα άλλο παράδειγμα είναι το paging σε λειτουργικά συστήματα όπου μια μικρή και γρήγορη μνήμη πρέπει να αποφασίσει ποια δεδομένα θα αντικαταστήσει με νεα ώστε να μπορεί να φέρει τα επόμενα δεδομένα που θα ζητήσει ο χρήστης πιο γρήγορα και να μη χρειαστεί να αναφερθούμε στην αργή και μεγάλη μνήμη.

Για την ανάλυση online αλγορίθμων ορίζουμε το competitive ratio, στο οποίο συγκρίνουμε το αποτέλεσμα του αλγορίθμου μας με τη βέλτιστη λύση του αλγορίθμου αν είχε όλα τα δεδομένα απ την αρχή.

1.2 Online Προβλήματα Χωροθέτησης

Σε αυτήν την ενότητα θα παρουσιάσουμε συνοπτικά τα σημαντικότερα αποτελέσματα σε Online προβλήματα Facility Location και μια παραλλαγή του, το Sum Radii, όπου τα facilities ανοίγουν με συγκεκριμένη ακτίνα και οι πελάτες δεν πληρώνουν κόστος σύνδεσης αλλά μπορούν να συνδεθούν σε ένα facility μόνο αν είναι εντος της ακτίνας του. Αρχικά αναφέρουμε ότι το lower bound που υπάρχει στο online Facility Location είναι $\frac{\log n}{\log \log n}$ και αποδείχθηκε στο [3].

Ο πρώτος που όρισε το online Facility Location ήταν ο Meyerson [4] ο οποίος παρουσίασε και έναν πολύ απλό και διαισθητικό randomized αλγόριθμο που πετυχαίνει $\log n$ προσέγγιση. Σε αυτόν τον αλγόριθμο κάθε φορά που έρχεται ένα πελάτης, έστω δ η απόσταση του από το κοντινότερο facility, τοτε θα ανοίξει στη θέση που έφτασε ένα facility με πιθανότητα δ/f όπου f το κόστος ανοίγματος του facility. Ουσιαστικά ο αλγόριθμος λέει ότι όταν είμαστε πολύ μακρυα από το κοντινότερο ανοιχτό facility, θα πρέπει η πιθανότητα με την οποία ανοίγει ένα νεο facility να είναι μεγάλη, ώστε και οι επόμενοι πελάτες που θα φτάσουν στην περιοχή να μη πληρώσουν πολλά. Στο [5] μια καλύτερη ανάλυση του αλγορίθμου έδειξε ότι στη πραγματικότητα είναι ασυμπτωτικά βέλτιστος καθώς φτάνει το $\frac{\log n}{\log \log n}$.

Στη συνέχεια υπήρξαν και άλλοι αλγόριθμοι, όπως στο [6], ο οποίος χρησιμοποιεί τη μέθοδο πρωτεύοντος - δυϊκού και την έννοια του "δυναμικού" μιας περιοχής. Το δυναμικό ειναι ουσιαστικά το πόσο πολύ θέλουν οι γύρω πελάτες να ανοίξει facility σε ένα συγκεκριμένο σημείο του χώρου. Οταν οι πελάτες κερδίζουν περισσότερο από το f τότε θα ανοίξει facility στο σημειο. Ένας ακόμα ενδαιαφέρον αλγόριθμος [7], ξεκινάει με ένα τετράγωνο με διαγώνιο f και όσο έρχονται νεοι πελάτες χωρίζει το τετράγωνο κάθε φορά σε 4 μικρότερα τετράγωνα οταν το κόστος σύνδεσης ξεπεράσει ένα όριο. Αυτό είναι ουσιαστικά παρόμοιο με την έννοια του δυναμικού. Και οι δύο αυτοί αλγόριθμοι πετυχαίνουν λογαριθμικό λόγο προσέγγισης.

Τελικά στο [3] δοθηκε ένας βέλτιστος ντετερμινιστικός αλγόριθμος, με σχετικά περίπλοκη ανάλυση, που δίνει ratio όσο το lower bound, και χρησιμοποιεί πάλι την έννοια του δυναμικού αλλά το σημείο που θα ανοίξει το νεο facility επιλέγεται πιο προσεκτικά.

Στο πρόβλημα Sum Radii, που περιγράφηκε παραπάνω, έχουν δοθεί δύο αλγόριθμοι, ένας ντετερμινιστικός και ένας randomized με λόγο προσέγγισης $\log n$ και οι δύο. Ο randomized για κάθε πελάτη που φτάνει και δεν καλύπτεται, προσπαθεί να ανοίξει ένα facility με ακτίνα $2^i f$ με πιθανότητα $1/2^i$. Στον ντετερμινιστικό, χρησιμοποιείται πάλι η μέθοδος πρωτεύοντος-δυϊκού.

1.3 Μια Νεα Παραλλαγή στο Πρόβλημα - Radii FL

Τέλος, ορίζουμε μια νέα παραλλαγή στο πρόβλημα την οποία ονομάζουμε Radii-Facility Location (Radii-FL για συντομία) η οποία είναι ουσιαστικά ο συνδυασμός δύο προβλημάτων που συζητήθηκαν πιο πριν: του Facility Location και του Sum Radii. Σε αυτο το πρόβλημα, υπάρχει ο περιορισμός των ακτινών, δηλαδη όταν ανοίγουμε το facility πρέπει να αποφασίσουμε με τι ακτίνα θα ανοίξει. Τελικά θα πληρώσουμε για κάθε ανοιχτό facility ένα κόστος ανοίγματος, ένα κόστος για την ακτίνα και το άθροισμα των αποστάσεων για να συνδέσουμε τους πελάτες. Το γραμμικό πρόγραμμα για το πρόβλημα φαίνεται στους παρακάτω πίνακες.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i,r} y_i^{(r)}(f + fr) + \sum_{i,j,r} x_{ij}^{(r)} f_3 d(i,j) & \\
\text{subject to} \quad & x_{ij}^{(r)} \leq y_i^{(r)} & ,\forall i,j,r \\
& \sum_{i,r} x_{ij}^{(r)} \geq 1 & ,\forall j \\
& x_{ij}^{(r)} \geq 0 & ,\forall i,j,r \\
& y_i^{(r)} \geq 0 & ,\forall i,r
\end{aligned}
$$

Table 1.4: Πρωτεύον, $i \in F$ and $j \in C$

Στην περίπτωση μας, προσθέσαμε και έναν συντελεστή στο κόστος σύνδεσης. Αυτό έγινε γιατί αν $f_3 = 1$ δε θα είχε νόημα το πρόβλημα: το κόστος σύνδεσης θα μέτραγε τόσο πολύ σε σχέση με το κόστος ακτίνας και το κόστος ανοίγματος, που πάντα θα μας συνέφερε να ανοίξουμε νεο facility με την ελάχιστη ακτίνα παρά να συνδέσουμε τον πελάτη με κάποιο ήδη ανοιχτό, πιθανότατα με μεγαλύτερη ακτίνα.

$$
\begin{aligned}
\text{maximize} \quad & \sum_{j \in C} b_j \\
\text{subject to} \quad & b_j \leq a_{ij}^{(r)} + f_3 d(i,j) && ,\forall j, i, r : d(i,j) \leq r \\
& \sum_{j \in C} a_{ij}^{(r)} \leq f + fr && ,\forall i, r \\
& a_{ij}^{(r)} \geq 0 && ,\forall i, j, r \\
& b_j \geq 0 && ,\forall j
\end{aligned}
$$

Table 1.5: Δυϊκό, $i \in F$ and $j \in C$

Αυτό σημαίνει ότι για διαφορετικές τιμές του συντελεστή f_3 θα πρέπει και ο αλγόριθμος να συμπεριφέρεται διαφορετικά. Δηλαδή, αν πούμε OPT την βέλτιστη offline λύση στο πρόβλημα, για μεγαλες τιμές του f_3 η OPT δε θα ανοίγει πολύ μεγάλα facilities, άρα δεν έχει νόημα και εμείς να ανοίγουμε μεγάλα. Ο αλγόριθμος κάνει ακριβώς αυτό, για πελάτες που δεν καλύπτονται θα προσπαθει να ανοίξει τοσο μεγάλα facilities όσο θα μπορούσε να ανοίξει και η OPT. Στην περίπτωση που ο πελάτης καλύπτεται, ουσιαστικά θα τρέχουμε τον αλγόριθμο του Meyerson. Ο αλγόριθμος φαίνεται αναλυτικά στο 1. Για λογους απλότητας και για να διευκολυνθούμε στην ανάλυση λέμε ότι $f_3 = 1/2^M$.

Algorithm 1: Ο αλγόριθμος FL_Radii

Data: M

New demand p:
if p *is covered by* c_i **then**
    $\delta = min\{d(i,p) | i \in F\}$;
    // 2^{u_i} `rad of facility covering` p
    $u = u_i - 1$;
    With $p_{cov} = \frac{\delta}{2^u f + f}$ open facility at p (cost: $2^u f + f$)
end
else
    // u_i `is not covered`
    With $p_{uncov} = \frac{1}{2^i}$, $\forall i \in [M]$ open facility at p (cost: $2^i f + f$)
end

Μελετώντας το competitive ratio, περιμένουμε να εξαρτάται από την παράμετρο M, κάτι που όντως συμβαίνει καθώς αποδεικνύουμε ότι έχουμε ratio: $\max\{\frac{\log n}{\log \log n}, \min\{\log n, M\}\}$.

Αυτό ουσιαστικα μας λέει, ότι για μικρές τιμές του M, αυτό που υπερισχύει στο competitive ratio είναι το $\frac{\log n}{\log \log n}$ που ουσιαστικά προκύπτει επειδή ο OPT αναγκάζεται να λύσει facility location μέσα στα μικρά clusters που ανοίγει. Σε αντίθετη περίπτωση, το M υπερισχύει του Facility Location, και το πρόβλημα τείνει να γίνει σαν το Sum Radii. Στην περίπτωση που το M είναι πολύ μεγάλο, άρα το κόστος σύνδεσης πολύ μικρό, ουσιαστικά έχουμε το Sum Radii.

Chapter 2

Introduction

The problem of finding a point that minimizes the sum of distances from a given set of points is known since the early 17th century. Since then, it has been studied in many different contexts, from economics and location theory, to operations research and lately to computer science, known as the Facility Location Problem. This is a classic problem in combinatorial optimization, and people soon realized it is hard to solve; it belongs to a class called $\boldsymbol{NP}$-Hard problems, so there is a chance that we will never be able to find a polynomial time algorithm for them.

In this case, we try to find a solution "close" to the optimal, and this is what we call an approximation algorithm: an algorithm with a mathematical guarantee of how "far from the optimal we are in the worst case. This led us to define the concept of *Approximation ratio*, which is defined as $c(n) = \max\{ALG/OPT, OPT/ALG\}$ where ALG is cost of the solution given by our algorithm, and OPT is the optimal solution. Using this definition, the approximation ratio is > 1 for both minimization and maximization problems. So if we prove that $c(n) \leq f(n)$ for any input of the algorithm, then we have a $f(n)$-approximation to the problem.

Usually, the traditional setting in algorithms is that we are given the input from the beginning, and then try to find a solution, or approximate it as we said before. However, this is not the case in real world problems, where parts of information of the input, or even the whole input is not available from the beginning. Consider for example that we want to divide some data into groups, according to their similarity. This could be medical data or consumer preferences for example. It is clear that in any context, algorithms for these problems will never have the whole input, but we should potentially make decisions based on the input given until now. Therefore, it comes as a natural "extension" of almost every problem to be considered in such an uncertain environment. This setting entails many different approaches on algorithms (*online*, *incremental*, or *dynamic*, *streaming*) depending on how we can change the solution in each step, or if we care about space or time complexity.

The tools and techniques used in the analysis of online and incremental algorithms, which are the ones we will discuss in this thesis, are very similar to those used in approximation algorithms. This happens because *competitive analysis*, which will be used throughout this thesis to measure the performance of an online algorithm, is essentially an extension of the notion of the approximation ratio to online algorithms. The *Competitive ratio* is the measure of performance for an online algorithm and is defined as: $c = C_{\mathcal{A}(\sigma)}/C_{\mathcal{OPT}(\sigma)}$ where σ is an input sequence to the online algorithm, $\mathcal{A}$ is the solution given by our algorithm and $\mathcal{OPT}$ is the optimal offline solution. So if we prove that $c \leq a$, we have an a-competitive algorithm.

In all these settings, we need a way to formally define these problems, in order to try to find mathematical guarantees for competitiveness or the approximation ratio. To do this we use Linear Programming, perhaps one of the most powerful tools in solving combinatorial optimization problems. A typical linear program consists of an objective function to be minimized, of the form $\boldsymbol{c}^T\boldsymbol{x}$ and constraints $\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq 0$. LP tells us that this comes as a pair with a *dual* maximization problem: $\max\{\boldsymbol{b}^T y | \boldsymbol{A}^T y \leq c, \boldsymbol{y} \geq 0\}$. The main concept of linear programming, that makes it so powerful, is that of duality. Duality tells us that the two aforementioned problems have the same optimal solution, and using this we can actually obtain guarantees of optimality for either the primal or the dual problem. Except for this, *complementary slackness*, which is derived from duality, gives us the connections between $\boldsymbol{x}$ and the dual constraints and vice versa, so exploiting this structure of the primal and dual programs we can find approximations to problems, without actually solving the LP. This is what the quite elegant primal-dual method essentially does: by maintaining a dual feasible solution, and satisfying approximate complementary slackness conditions, we get the approximation guarantee. Other typical techniques involve rounding, where we first solve the LP, and then try to round the solution to an integer one, either deterministically or using randomization.

Facility Location

Facility Location is a very naturally motivated problem, and this is why it has been studied in so many settings. For example, imagine you are the mayor of a city which has no fire stations yet (a dangerous city to live in!). As a good mayor, you want the residents to feel safe, so you want to build fire stations throughout the city, in places where in case of fire, there will be a fire station relatively close. Of course we could just build a fire station in each road, but unfortunately the city's budget is limited. So, there is a cost to open a fire station, and the "sum of distances" metric is a very natural way to measure how good a placement of the station will be. Since the city does not want to pay much, we want to minimize the opening costs, and the sum of distances from the closest station.

Formally, we are given a set of potential facility locations F, a set of clients C, a function $d : F \times C \to \mathbb{R}$ (usually a metric) and opening costs f_i for each $i \in F$. We

want to open a subset of facilities $F' \subseteq F$ (by paying an opening cost for each one) and connect every client to an open facility in such a way as to minimize the sum of opening and connection costs: *minimize* $\sum_{i \in F'} f_i + \sum_{i \in F', j \in C} d_{ij}$.

This problem however proved to be $\boldsymbol{NP}$-Hard and in [2] Guha and Khuller showed a lower bound of 1.463 on the approximation ratio, assuming $\boldsymbol{NP} \notin \boldsymbol{DTIME}[n^{O(\log \log n)}]$.

Given the hardness of the problem, computer scientists tried to tackle it using many different approaches, but perhaps the more interesting ones - giving the most elegant algorithms- are those using linear programming. Linear programs provide a very powerful tool in optimization, mainly because of strong duality. Duality gives us far better understanding of the variables and the constraints of the problem, especially when the dual has a nice intuitive interpretation, as in the case of Facility Location and generally covering/packing problems. This information the dual gives us, can be used to guide us in the search for the solution, as for example in the primal - dual schema which utilizes the concept of complementary slackness - a direct implication of strong duality.

Currently the best approximation algorithm for Uncapacitated Facility Location gives a 1.488-approximation, proved by Li in [1], using a modification of Byrka's 1.5 - approximation algorithm in [8]. However, since the proof of the lower bound in 1998, many different approaches were tried on the problem, giving different approximations ; among them were LP rounding ([9], [10], [11]), the primal dual schema ([12]), greedy approaches some using the factor revealing LP ([13], [14]) or a combination of some of the above ([15], [2], [16]). Some of these techniques are presented in chapter 4 where we discuss two simple LP rounding algorithms, and most importantly the first primal-dual algorithm for Facility Location given by Jain and Vazirani.

As we discussed previously though, it is not always a realistic scenario that we know all the clients/demands right from the beginning. Consider the fire stations problem ; new residents arrive as the city grows, and they also want to be close to a fire station. Towards this direction, Meyerson [4] first introduced the online Facility Location, where the demands arrive one by one, and we must decide whether to assign them to an already open facility, or open a new one, giving a quite simple and intuitive randomized algorithm. Since then, there is much work done in this problem, most notably Fotakis in [3] who gave a lower bound of $\frac{\log n}{\log \log n}$, showed [5] that Meyerson's algorithm is asymptotically optimal, achieving this lower bound, and gave a more complicated optimal deterministic algorithm. A simpler and more intuitive deterministic algorithm, giving a logarithmic competitive ratio was given in [6]. The interesting element of this algorithm is that it defines the notion of the *potential* of an area, which essentially quantifies how much an area will "benefit" from the opening of a facility there. The interesting thing is that this is exactly what the LP of the problem tells us should happen; the potential should not exceed the facility cost in the area. Another interesting algorithm, again using the notion of the potential, was given by Anagnostopoulos et al. [7], achieving a logarithmic competitive ratio. In this case, they gave an algorithm for the plane, implicitly using binary search, to find the best facility position, using the potential as a guide.

It is worth noting that the optimal deterministic algorithm of [3] is quite complicated to analyze, since it needs to exploit the potential and the "area" around demands in a non trivial way, compared to [7] and [6], but on the other hand, it is very easy and intuitive to get an optimal algorithm using randomization, as in [4], where the analysis is also straightforward.

Sum Radii

Imagine now, you are the Dean of an anonymous university, where there are many labs and teaching rooms, where we want people sitting there to have access to the university's wifi. Therefore, we need to be sure, that the wifi repeaters, that consume different power each and have analogously different range, will be placed in the best possible positions, so that everyone will have access. Since (as always) the university's budget is limited, we cannot give everyone a wifi repeater of their own, so we need to place different ones on different places throughout the university campus, to satisfy everyone, while paying the less possible.

Formally in this problem, we are given a set of clients C, and a function $d : C \times C \to \mathbb{R}$. We want to open k clusters from the set C, lets say C', in order to minimize the sum of cluster radii, where the radius of a cluster K is $r_i = max_{i \in C', j \in C \setminus C'} d(i, j)$. So we want to minimize $\sum_{i \in C'}(r_i)$.

For this problem, in [17] they give a $\log(n/k)$ using at most $10k$ clusters, while in the more recent [18] they first find a 3 approximation for the problem without restriction to the number of clusters, and a 3.504 approximation randomized algorithm using at most k clusters.

As we discussed in the Facility Location problem before, knowing all the demands from the beginning is not usually the case in the real world. Consider the university wifi problem from before, when the university grows, and new buildings and labs are built, we need to cover them too with wifi (or the researchers there will get very angry with the Dean).

In this online setting, we usually consider the Facility Location-like objective, where we drop the "at most k clusters" restriction and we want to *minimize* $\sum_{i \in C'}(f + r_i)$, where f is an "opening" cost for the center of each cluster. The demands arrive one by one, and must be irrevocably assigned to a cluster or open a new facility with a radius fixed at opening time.

In this direction, Fotakis and Koutris in [19] gave a primal-dual deterministic and a randomized algorithm that both give logarithmic competitive ratio, while proving a logarithmic lower bound on deterministic algorithms and a $\log \log n$ lower bound on randomized algorithms. In a slightly different setting, where we are allowed to make changes to the current solution i.e. close an open facility, or "merge" two opened facilities, Fotakis in [20] gave a constant-competitive algorithm again using the notion of potential (as in

[3]) to open facilities, but ensuring with the merge rule, that essentially the location of the facility converges to the optimal one.

In the incremental setting, where we also have the "at most k clusters" restriction, Charikar and Panigrahy in [18] gave a deterministic algorithm which uses the primal dual schema and achieves a 160-approximation using at most $4k$ clusters.

Other Related Problems

There are many problems closely related to Facility Location and Sum Radii. A problem very close to Uncapacitated Facility Location is the k-median problem, where the objective is to divide the data into k clusters in order to minimize the sum of connection costs. In this direction, in the offline metric setting, Lin and Vitter in [21] proved a $2(1 + 1/\epsilon)$-approximation using at most $(1 + \epsilon)$ centers. Later, Charikar et al in [22] gave a 6 2/3-approximation, and Jain-Vazirani in [12] showed a 6-approximation for the k-median problem using as a building block a 3 approximation algorithm for the Facility Location problem, proved earlier in the paper. The problem was first introduced to the online setting by Mettu and Plaxton [23] who also gave a constant-competitive algorithm using a recursively greedy criterion, in order to avoid fixing the location of the center too early (and yield an unbounded competitive ratio). In the incremental setting, where we can also merge clusters, Charikar and Panigrahy [18] provide a constant competitive algorithm using $O(k)$ centers, but assuming that we know the number of demands from the beginning. It is worth noting, that in this paper the technique used is the primal-dual, which essentially tries using binary search to find the best "size" of the clusters. Later, Fotakis [20] proved a constant competitive algorithm using again $O(k)$ enters, without this assumption.

Another variant, closer to Sum Radii, is the k-center problem, where we want to minimize the maximum radius of a cluster. Gonzalez in [24] proved a greedy 2-approximation algorithm, while more recently, Badoiu et al. in [25] showed a $(1 + \epsilon)$-approximation for the Euclidean space. Motivated by problems in information retrieval, Charikar et al. [26] defined the incremental version of this problem where demands arrive online, but we can also merge two existing clusters. They provide various deterministic and randomized algorithms, achieving constant approximation ratios to the problem. This is very closely related to clustering algorithms, especially agglomerative clustering, where we start with many points, and try to merge them into k clusters (see [27] for a survey on hierarchical clustering algorithms).

Radii Facility Location

Imagine now, that in a PC lab we wanted to cover with wifi repeaters, we also want an ethernet cable to reach every seat. This would mean that apart from the wifi repeater, which we will pay according to how much power it consumes, also the cable should be paid.

This new variant, which we call *Radii Facility Location* (Radii FL for short), addresses exactly that problem in the online setting; demands arrive one by one, and must be irrevocably assigned to an already open facility, which they can only do if the are inside its radius, or decide to open a new facility with a radius fixed at opening time. Formally, we are given a set of points C and a distance function $d : C \times C \to \mathbb{R}$. We need to open a set $F' \subseteq C$ of facilities, with radii $r_i,\ i \in F'$, and assign each client to a facility in F'. The objective in this case is to minimize $\sum_{i \in F'}(f + r_i) + \sum_{j \in C, i \in F'} f_3 d(i, j)$. f_3 is a scale factor that essentially says how expensive is the connection cost.

This variant is the main contribution of this thesis, and is presented in more detail in chapter 6. Specifically, we explain the role of the scale factor for our problem, the reasons why it is necessary and how the problem changes for different values of it; when $f_3 = 0$ for example, it is easy to see that the problem reduces to Sum Radii, while for large values of f_3, when the assignment cost is more "expensive", the problem essentially becomes Facility Location. Following this, we design a randomized algorithm, naturally depending on this factor, by combining the two simple randomized algorithms for Sum Radii and Facility Location presented in chapter 5 of the thesis. This is quite natural, since the problem changes from Facility Location to Sum Radii for different values of f_3, so we just needed to incorporate this in our combination. Since the algorithm depends on f_3, we also expect such dependency to exist in the competitive ratio, which naturally does; we prove the competitive ratio to be $\max\{\frac{\log n}{\log \log n}, \min\{\log n, M\}\}$ where $f_3 = 1/2^M$ for simplicity. This shows clearly that for small values of f_3 the $\log n$ factor coming from Sum Radii dominates the competitive ratio, while for larger values of f_3 the $\frac{\log n}{\log \log n}$ factor originating from the Facility Location problem is larger. This essentially tell us where the difficulty of the problem lies in each case.

Chapters Overview

In chapter 3 we give a brief introduction to the tools we will use later in the thesis in the analysis and design of the algorithms. The basic one is linear programming, mainly duality and complementary slackness, presented in section 3.1. After this, we introduce the basic notions of approximation algorithms and competitive analysis, used for the online algorithms.

In chapter 4 we introduce the reader to the classic offline Facility Location problem

and the previous work done in this direction. We also give three algorithms, each one using a different design technique used generally in approximation algorithms; deterministic rounding, randomized rounding and primal - dual schema.

In chapter 5, we formulate the online version of Facility Location alongside the Sum Radii variants and present in more detail some of the most important previous work done in terms of algorithms and lower bounds for these problems.

Finally in chapter 6 where we formally define the *Radii Facility Location* variant, and discuss how the scale factor f_3 affects the optimal solution. Following this, we give an algorithm, based on this scale factor and discuss how the competitive ratio of this algorithm changes for different values of f_3.

Chapter 3

Preliminaries

In this section we present some basic tools that are used in the analysis and design of the algorithms presented later in this thesis. We start by introducing the basic concepts of Linear Programming that are used to formulate and solve algorithmic problems. In section 3.2 we present the basic concepts used in approximation algorithms, while in section 3.3 we introduce the reader to online algorithms and to the well known problems in the area. Finally, in 3.4 we define some basic facts about metric spaces and Hierarchically Well Separated trees, in order to use them later in the thesis.

3.1 Linear Programming

A linear program is the problem of optimizing a linear objective function subject to inequality or equality constraints. The standard form LP is shown below. Typically this is the called the Primal and comes as a pair with a Dual, which is shown in table 3.2.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i & \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} x_i \geq b_j & \forall j \in [m] \\ & x_i \geq 0 & \forall i \in [n] \end{aligned}$$

Table 3.1: Canonical form of Primal problem (P)

The intersection of the constraints of the program above, define a polyhedron in $\mathbb{R}^n$. For a point $\boldsymbol{x} \in \mathbb{R}^n$ to be called a *feasible solution*, $\boldsymbol{x}$ needs to satisfy all the constraints of

the problem. If additionally $\boldsymbol{x}^* = \min \boldsymbol{c}^T \boldsymbol{x}$, then $\boldsymbol{x}^*$ is called an *optimal solution*. If there is at least one $\boldsymbol{x}$ that is a feasible solution, then the LP is *feasible*.

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} b_j y_j \\ \text{subject to} \quad & \sum_{j=1}^{m} a_{ij} y_j \leq c_i \quad \forall i \in [n] \\ & y_j \geq 0 \qquad\qquad \forall j \in [m] \end{aligned}$$

Table 3.2: Canonical form of Dual problem (D)

There are many algorithms for solving linear programs. The first one was proposed by Dantzig in 1947, which is exponential in the worst case. In 1980 however, LP was proved to be in P [28] by Khachiyan who proposed the ellipsoid algorithm. And finally, in 1984 Karmarkar [29] introduced another poly-time algorithm, which is better in practice than ellipsoid.

3.1.1 Duality

Duality is maybe the most important concept in Linear Programming since it allows us to give proof of optimality for a solution. Weak duality says essentially that the dual maximization problem is a lower bound for the primal minimization problem. Strong duality, says that the optimal solution is the same for both primal and dual problems, so they are essentially trying to reach the same point, from different directions.

Theorem 3.1.1 (Weak Duality)**.** *Let x, y be feasible solutions for the primal and the dual respectively. Then:* $\sum_{i=1}^{n} c_i x_i \geq \sum_{j=1}^{m} b_j y_j$

Theorem 3.1.2 (Strong Duality)**.** *The primal is feasible iff the dual has a finite optimal solution. In this case:* $\sum_{i=1}^{n} c_i x_i = \sum_{j=1}^{m} b_j y_j$

Weak duality is easily proved using the fact that x, y are primal and dual feasible so they satisfy the respective constraints, while for the strong duality we need Farkas' Lemma.

3.1.2 Complementary Slackness

Another important concept, used heavily in the design of algorithms, especially in the primal-dual schema is that of complementary slackness. This reveals the relation between the variables of the primal and the dual constraints and vice versa. Specifically, it says that when a primal variable is strictly greater than zero, the respective dual constraint must be tight- meaning it is satisfied with equality.

Definition 3.1.1 (Complementary slackness)**.** $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ are optimal solutions for (P) and (D) respectively iff:

- *Primal CS*: $x_i^* \Big(\sum_{j=1}^{m} a_{ij} y_i^* - c_i \Big) = 0, \ \forall i \in [n]$
- *Dual CS*: $y_i^* \Big(\sum_{j=1}^{m} a_{ij} x_j^* - b_j \Big) = 0, \ \forall j \in [m]$

We could say, that each dual variable that corresponds to a primal constraint, indicates how much this constraint "matters" to the solution. In an optimal solution x^* there are n constraints satisfied with equality, since x^* is a vertex of the polytope. This essentially says, that if a constraint "matters" for the solution, i.e. has positive "weight" it should be tight, (satisfied with equality).

There are many cases where we relax the complementary slackness conditions in order to prove an approximation factor. Specifically we have the approximate primal and dual complementary slackness conditions:

- *Primal*: if $x_i > 0$ then for $\alpha \geq 1$, $\forall i \geq 0$ we have $\frac{c_i}{\alpha} \leq \sum_{j=1}^{m} a_{ij} y_j \leq c_i$
- *Dual*: if $y_j > 0$ then for $\beta \geq 1$, $\forall j \geq 0$ we have $b_i \leq \sum_{i=1}^{n} a_{ij} x_j \leq \beta \cdot b_j$

It is easy to see that if we have two solutions to the primal and dual programs that satisfy the approximate complementary slackness conditions, we get $\sum_{i=1}^{n} c_i x_i \leq \alpha\beta \sum_{j=1}^{m} b_j y_j$. This means that if we find primal and dual solutions that satisfy approximate complementary slackness, then we have a $\alpha\beta$-approximation to the optimal.

3.1.3 Integer Programming

Linear programs usually find fractional solutions, meaning that variables $x_i \in \mathbb{R}$. When modeling real life problems, this may not exactly reflect reality, and usually we need

the variables to be natural numbers. Similarly to 3.1, if we substitute the non negativity constraints with $x_i \in \mathbb{Z}_+^n$ we get an integer program. Typically, the variables are $x_i \in \{0, 1\}$ since they represent a yes/no decision. For example in the Set Cover problem, we define variables: $x_i = 1$ iff we choose set i in the final solution and $x_i = 0$ otherwise.

However, since Integer Programming is ***NP***-Hard, we usually get the LP of table 3.1 as the *relaxation* of the problem, and solve a linear program instead of an integer one. We should be careful when we relax a program though, since the fractional optimal will be less than the integer optimal. This is easy to see, since the fractional allows us more freedom in the choice of the variables. This is quantified as the *integrality gap* between the Integer and the Linear programs. Formally integrality gap is defined as the ratio $Frac/Int$ for minimization problems, where Int and $Frac$ are the optimal Integer and fractional solutions respectively.

For a brief introduction to Linear Programming and the basic algorithms see [30] and for a more in depth view with some applications to problems see [31].

3.2 Approximation Algorithms

Sometimes we are required to find a minimum or maximum solution to a problem - in short to solve an optimization problem- for which there is no fast algorithm known to be good in any practical setting. These problems are usually ***NP***-Complete or even ***NP***-Hard, which means that possibly we will never be able to find an algorithm that runs in less than exponential time. In cases when the instances are small, exponential time may not be a problem, or sometimes there are special cases of a problem, that arise from real life and are actually easy to solve, these usually do not occur however. This leads us to the natural thought: maybe if we relax the "exact solution" constraint, we could obtain algorithms that run in reasonable time to be useful to applications since many real life problems are ***NP***-Hard.

The target of the design of an approximation algorithm, is to give a mathematical guarantee of how close the approximate solution will be to the optimal for each instance. More specifically, we give the definition of a c approximate algorithm and of the approximation ratio, which is the central concept in the field.

Definition 3.2.1. A $c(n)$-approximation algorithm for an optimization problem is a polynomial- time algorithm that for all instances of the problem produces a solution whose value is within a factor of $c(n)$ of the value of an optimal solution, where n is the size of the input.

The factor $c(n)$ is essentially the performance guarantee of the algorithm, it is called the *approximation ratio* and is usually defined to be $c \geq 1$.

Therefore, for a minimization problem we have that $ALG \leq c(n) \cdot OPT$ and for a maximization problem $ALG \geq c(n) \cdot OPT$ where ALG and OPT are the values of the solutions of our algorithm and the optimal solution respectively.

3.2.1 Approximation Schemes

There are some problems, which we can approximate as much as we want, by "paying" more in computation time the closer we get to the optimal. These algorithms are called *Approximation Schemes* and require as input, except for the input of the problem, a number $\epsilon > 0$ which tells the algorithm how close we want to get to the optimal solution- essentially the approximation ratio we require the algorithm to have.

In such algorithms, the running time is polynomial in n, but the running time involves the parameter ϵ, for example $O(n^{1/\epsilon})$, and they are called *Polynomial Time Approximation Schemes* or *PTAS*. In the case where the running time is also polynomial in ϵ apart from n the algorithm is called *Fully Polynomial Time Approximation Scheme* or *FPTAS*; for example an algorithm running in $O((1/\epsilon)^3 n^4)$ would be an FPTAS.

For an $\boldsymbol{NP}$-Hard problem, an approximation scheme is the best we can hope for. In the process of designing approximation algorithms, we actually achieve a better understanding of the real difficulty of a problem ; set cover for example, is even hard to approximate within a constant factor: we cannot find a better than $c \log n$ approximation algorithm, for $c < 1/4$ unless $\boldsymbol{NP} \subseteq \boldsymbol{DTIME}(n^{poly \log n})$ [32].

For an excellent introduction to approximation algorithms see [33], or the slightly more advanced [34]. For an even more advanced approach see [35].

3.3 Online Algorithms

The traditional approach on algorithm design is to assume that the entire input is known to the algorithm beforehand. This is not realistic for many practical settings, where we learn parts of the input while the algorithm is running, and we should make decisions on the fly, while new data arrive. We present some examples of real life problems that require this approach, shown in [36]. In section 3.3.3 we present some examples of online algorithms in more detail. For a more detailed introduction to this area, see the book of Borodin and El-Yaviv [37].

- *Data Structures*: In this case, we have a data structure, for example a tree, on which we perform some operations such as insertion, search, deletion etc, which we need to be done fast. This is used in databases for example, where the data we will search or insert is not known in advance. An example for this is the list update problem.

- *Scheduling*: Job scheduling on processors is also a well known problem. The jobs arrive one by one, so their number and duration is not known beforehand, and we must decide when to schedule them. In some settings, the number of available machines/processors is different in each step - when a machine breaks down and cannot service any job.

- *Networks*: In networks many times, we need to maintain a set of open servers in order to service a set of clients/nodes of the network. We need to service every client the moment he arrives, by paying something towards maintaining the servers open and towards the power needed to service a client. In this case, there can be uncertainty in the number and location of clients that arrive or the servers that are capable of opening at each given time.

- *Resource management in Operating Systems*: Paging is a classic problem in operating systems. Usually there are two types of memory, cache memory, which is small in capacity, but fast, and Random Access Memory which is slower but with larger capacity. It is easier to retrieve data from cache memory than RAM, so when the user requests something, and it is not in the cache, the operating system must decide which data to evict from the cache, in order to be faster in the future. Obviously, this depends on the user's decision on which data to request, which cannot be known in advance.

In order to analyze the algorithms running on such settings, Sleator and Tarjan in [38] introduced the notion of competitive analysis where the performance of the online algorithm is compared to that of the optimal offline i.e. the algorithm that knew the entire input data from the beginning. Competitive analysis is in a sense worst-case analysis for online algorithms since we keep the worst possible output of our algorithm compared to the best possible offline algorithm's output. Additionally we assume total absence of information regarding the input data ; we could assume some distribution on the input data, which is something that would possibly improve the performance of our algorithm, but can be crude sometimes and not exactly reflect the reality. A better way, in order to get more accurate bounds on the algorithms is to use amortized analysis, introduced by Tarjan in [39]. In this case we essentially analyze more carefully the decisions made by the algorithm and how bad they can actually be ; for example a very bad decision now, may not allow us other bad decisions later.

Formally, an online algorithm $\mathcal{A}$ is presented with a request sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$. The requests $\sigma(t),\ 1 \leq t \leq m$ must be served in order of occurrence. When serving $\sigma(t)$ the algorithm does not know $\sigma(t'),\ t' > t$.

Definition 3.3.1 (Competitiveness)**.** Let $C_{\mathcal{A}(\sigma)}$ denote the cost of the algorithm and $C_{OPT(\sigma)}$ the cost of the optimal offline algorithm. Algorithm $\mathcal{A}$ is called *c-competitive* if

there exists a constant a such that $C_{\mathcal{A}(\sigma)} \leq c \cdot C_{OPT(\sigma)} + a$. If $a \leq 0$ we say that $\mathcal{A}$ is *strictly c-competitive*.

In the case when the algorithm is randomized, we define the competitive ratio against the oblivious adversary, and the only difference in the definition is that instead of $C_{\mathcal{A}(\sigma)}$ we have the expected cost: $\mathbb{E}[C_{\mathcal{A}(\sigma)}]$.

Note that the competitive ratio is essentially an extension of the approximation ratio for offline algorithms ; a strictly c competitive algorithm is essentially a c approximation algorithm with the restriction that the algorithm must make decisions online.

Typically, in online algorithms the decisions made are irrevocable, or in some cases we are allowed constant number of changes. For example when we decide to open a server to service some demands, we cannot close it in later time. However there are many different approaches to uncertainty in algorithms. Another popular one is dynamic algorithms, where we are mostly concerned with the running time of the algorithm. On the other hand, in streaming algorithms, we place a limit on the space the algorithm uses.

3.3.1 Adversary Models

The competitive ratio of any online algorithm is defined with respect to an adversary. There are three main adversary models, depending on how much freedom the adversary has when he generates the request sequence, ranging from the *oblivious adversary* -the weakest one- to the *adaptive offline* - the strongest one.

- *Oblivious Adversary*: This type of adversary even though he knows the description of the online algorithm, must fix the request sequence before the algorithm starts to process demands.
- *Adaptive Online Adversary*: In this case, the adversary generates the next request in each step, after having seen the algorithm's decisions in all the previous steps, and must serve it immediately.
- *Adaptive Offline Adversary*: This adversary knows everything, every decision of the algorithm, even the random number generator. This adversary is so strong that randomization does not help against him.

It is easy to see that the oblivious and the adaptive online adversary are equivalent for any deterministic algorithm, since the algorithm's answers are completely predictable. The following two theorems proved in [40] show the relation between the different types of adversaries.

Theorem 3.3.1. *If there is a randomized algorithm that is α-competitive against any adaptive offline adversary, then there also exists an α-competitive deterministic algorithm*

Theorem 3.3.2. *If G is a c-competitive randomized algorithm against any adaptive online adversary, and there is a randomized d-competitive algorithm against any oblivious adversary, then G is a randomized $(c \cdot d)$-competitive algorithm against any adaptive offline adversary.*

3.3.2 Lower Bounds: Yao's Principle

Yao in [41] studied the expected running time of algorithms from two different point of views, namely the *distributional* and the *randomized* approach. In the first one, some distribution is assumed for the input data, and we try to find fast algorithms under this assumption. In the second approach, we make no assumptions for the input, but we allow the algorithm to make random decisions. This led to the definitions of two different complexities: distributional and randomized for a problem $\mathcal{P}$, whose definitions are presented below.

Let R be a randomized algorithm, and $\mathbb{E}[cost(R, x)]$ is the expected cost paid by algorithm R when the input is x. Randomized complexity is defined as follows:

Definition 3.3.2 (Randomized complexity)**.** $F_1(\mathcal{P}) = \inf_R \max_x \mathbb{E}[cost(R, x)]$

This essentially is the best randomized algorithm on its worst input. As for the distributional complexity, let D be an input distribution, A a deterministic algorithm and $cost(A, D)$ the expected cost paid by the algorithm for input coming from distribution D. The distributional complexity is defined as follows:

Definition 3.3.3 (Distributional complexity)**.** $F_2(\mathcal{P}) = \sup_D \min_{A \in \mathcal{A}} cost(A, D)$

This is essentially the best deterministic algorithm for a fixed distribution. Yao's principle says that these two complexities are equivalent.

Theorem 3.3.3 (Yao's Principle)**.** $F_1(\mathcal{P}) = F_2(\mathcal{P})$

So, for any specific randomized algorithm R we have that:

$$\max_x \mathbb{E}[cost(R, x)] \geq \inf_R \max_x \mathbb{E}[cost(R, x)] \Rightarrow \max_x \mathbb{E}[cost(R, x)] \geq \sup_D \min_{A \in \mathcal{A}} cost(A, D)$$

So, when we want to derive a lower bound on our randomized algorithm, we just pick a "difficult" distribution, and show that any deterministic algorithm A has a high cost. This technique is used in [3] to prove the lower bound on online Facility Location, as we will see in a later section.

3.3.3 Examples of Online Algorithms

In this section, we will briefly mention two representative online problems. The list updating problem, which was one of the first online problems to be defined, and the k-server problem where some interesting results were shown.

3.3.3.1 Data Structures - The List Update Problem

The List Update problem was historically one of the first online problems to be defined. It was first proposed in [38] alongside the notion of competitive ratio. In this problem, we have a linked list, which we can only traverse linearly starting from the first one. A series of demands arrive, where each one requires to access an item in the list.

The total cost paid for each request, is the number of elements in the list we traverse in order to find the item specified by the demand. After serving the request, we can move the item in any position earlier in the list, with no extra cost. Additionally, we can swap two items on the list, any time, with a cost of 1.

For this problem there are some very simple and intuitive deterministic algorithms ; *Move To Front* where every accessed item is moved to the front of the list, and *Transpose* where we just swap the accessed item with the previous on the list. We should note here, that the simplest idea i.e. the *Move to Front* algorithm, is 2-competitive which is also the lower bound for deterministic algorithms for this problem.

The lower bound for randomized algorithms for this problem is 1.5 proved in [42]. In [43] Albers et al. provided a 1.6−competitive randomized algorithm, which is a combination of two randomized algorithms: *BIT* and *TIMESTAMP*. This is the best one known until now.

3.3.3.2 The k-server Problem

k server is perhaps the most famous online problem. It was first formulated by Manasse and McGeoch in [44]. Specifically, we have a graph with n nodes, and distances d_{ij} between edges that satisfy the triangle inequality. The graph G and the number of servers k is given to us on the beginning while requests arrive online. We can see the graph as a metric space. Each request is a node of the graph and to service it there should be a server one that node. If we need to transport one to service the demand, we will pay the distance traveled. Note that we can move more than one server, and in this case we will pay the sum of the distances covered. Each demand should become satisfied on arrival time, before any of the next demands are satisfied.

Manasse and McGeoch found a lower bound of k for deterministic algorithms on symmetric metric spaces, when they first introduced the problem in [44]. Later, Koutsoupias and Papadimitriou in [45] came close to this lower bound by proving $2(k-1)$ competitiveness using the work of Chrobak and Larmore [46] on the work-function algorithm.

The lower bound for randomized algorithm was shown in [47] to be $\Omega(\log n)$. In a very recent work [48] which is yet to be published, Lee found an $O((\log n)^6)$-competitive randomized algorithm for the k-server problem on any metric space, improving thus Papadimitriou's and Koutsoupias' previous result.

At this point, it is worth mentioning the *Metrical Task Systems*; a *Metrical Task System* is a pair (S, d): a set of states S and a metric d on the elements of S. The input to the algorithm is a sequence of tasks where each one needs to be processed. The algorithm each time, pays the transition from the previous state and the processing cost. *Metrical Task Systems* were introduced by Borodin et al. in [49], and generalize many online problems, such as k-server, paging and the list updating problem. In this work, Borodin et al. also proved a tight bound $2n - 1$ for deterministic online algorithms' competitive ratio, where n is the number of states.

3.4 Metric Spaces and HSTs

The more interesting variants of Facility location are usually defined in a metric space. We will present the basic definitions and some properties along with a construction in metric spaces called Hierarchically Well Separated tree that we will use in later section for the construction of lower bounds.

Definition 3.4.1 (Metric Space)**.** Let X be a set and d a real function d:$X \times X \to \mathbb{R}$. Then d is called a metric on X iff $\forall a, b, c \in X$:

- *positive property* $d(a, b) \geq 0$, with equality iff $a = b$
- *symmetric property* $d(a, b) = d(b, a)$
- *triangle inequality* $d(a, b) \leq d(b, c) + d(c, b)$

We say that $d(a, b)$ is the distance between a and b with respect to the metric d.

We usually write (X, d) is a metric space to specify both the set of points and the metric on it.

Next, we define k-Hierarchically Well Separated Trees - HSTs for short. They are simply complete k-ary trees of height h, such that the distance from the root to its children is D and on every path from root to leaf the distance drops by a factor of m on every level. So the distance from node v of level j to its children is D/m^j. If T_v is a subtree rooted at v, every point $z \in T_v$ satisfies these two properties:

- For $u \in T_v$: $d(z, u) \leq \sum_{i=j}^{h} \frac{D}{m^i} = \frac{1}{m^{j-1}(m-1)} \approx \frac{D}{m^j}$

- For all $u \notin T_v$: $d(z,u) \geq \frac{D}{m^{j-1}}$

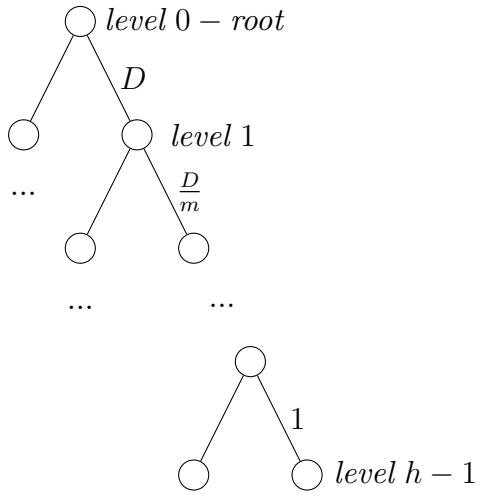

Figure 3.1: A binary HST

Chapter 4

Facility Location

In this chapter, we present three algorithms for offline Facility Location, each one using a different technique based on linear programming. These three techniques are widely used to design approximation algorithms for many different problems. After formulating the problem as an LP, in sections 4.2 and 4.3 we see deterministic and randomized rounding as presented in [33], while in section 4.4 we discuss the quite elegant primal - dual method through an algorithm first presented in [12] which is the technique we will mostly use in the subsequent chapters, for online Facility Location algorithms.

We discuss the simplest variant of Facility Location, namely the *Uncapacitated Facility Location*. We are given two sets of points: F-the set of potential facilities, and D-the set of demands/clients. Additionally $\forall i \in F$ and $\forall j \in D$ there are numbers $c_{ij} \geq 0$ that represent the cost of connecting client i to facility j and $\forall i \in F$, we are given a number f_i which is the cost of opening a facility at point i. The goal is to choose a set $F' \subseteq F$ of facilities to open, and connect every client to an open facility while trying to minimize the total opening cost and assignment cost, namely $\sum_{i \in F'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij}$. It is important to note that in this problem, c_{ij} is a metric. There is also the non-metric variant of this problem (*non-metric Facility Location*).

4.1 LP Formulation

We can write the Facility Location problem as an integer program with variables $y_i, x_{ij} \in \{0, 1\}$ where $i \in F, j \in D$. $y_i = 1$ iff facility i is open, and $x_{ij} = 1$ iff client j is connected to facility i else they are 0. For the reasons explained in section 3.1.3, we will work with the LP relaxation of the problem and its dual, shown in table 4.1.

minimize	$\sum_i y_i f_i + \sum_{i,j} x_{ij} c_{ij}$		maximize	$\sum_j a_j$	
subject to	$\sum_i x_{ij} \geq 1$	$, \forall j$	subject to	$a_j - b_{ij} \leq c_{ij}$	$, \forall j, i$
	$x_{ij} \leq y_i$	$, \forall i, j$		$\sum_j b_{ij} \leq f_i$	$, \forall i$
	$x_{ij} \geq 0$			$a_j \geq 0$	
	$y_i \geq 0$			$b_{ij} \geq 0$	

Table 4.1: LP formulation of the problem, $i \in F$ and $j \in C$ for the above

4.2 Deterministic Rounding

In this section we present a deterministic rounding algorithm for Facility Location. Say we have formulated the linear relaxation of the problem, and solved it. This will almost always give us a *fractional* solution. But what does it mean for a facility to be 0.3 open? In deterministic rounding, we try to recover a solution to the problem that makes sense for our setting, i.e. the variables should be integers. Since the fractional solution will be less than the integer, we want to round in a way such that the additional cost will not be very much.

We define the neighborhood for each client j to be: $N(j) = \{i \in F : x_{ij} > 0\}$. This is essentially the facilities that client j is fractionally connected to in the optimal LP solution. We also define the augmented neighborhood of each client j: $N^2(j) = \{l \in D : \exists i \in N(j) \wedge i \in N(l)\}$, which is essentially the clients that are connected to a facility in j's neighborhood.

Algorithm 2: Deterministic rounding for FL

Data: Primal and dual optimal solutions (x^*, y^*), (a^*, b^*)

$temp \leftarrow D$;
$k \leftarrow 0$;
while $temp \neq \emptyset$ **do**
  $k \leftarrow k + 1$;
  $j_k = argmin_{j \in temp}\{a_j^*\}$;
  $i_k = argmin_{i \in N(j_k)} f_i$;
  open i_k;
  assign j_k and $\{j \in N^2(j_k) : j$ is unassigned$\}$, to i_k;
  $temp \leftarrow temp - \{j_k\} - N^2(j_k)$;
end

Note that the neighborhoods form a partition of the facilities, because in each round, we assign every client $l \in N^2(j)$ that is unassigned, and therefore no facility of $N(j_k)$ can be chosen in a subsequent round.

Theorem 4.2.1. *This algorithm achieves an approximation factor of* 4 *for Uncapacitated Facility Location.*

Lemma 4.2.2. $\sum_k f_{i_k} \leq \sum_{i \in F} f_i y_i^* \leq OPT$

Proof. We bound f_{i_k} and then sum for all k.

$$f_{i_k} = f_{i_k} \sum_{i \in N(j_k)} x^*_{ij_k} \leq \sum_{i \in N(j_k)} f_i x^*_{ij} \leq \sum_{i \in N(j_k)} y^*_i f_i$$

Where the equality is due to $\sum_{i \in N(j_k)} x^*_{ij_k} = 1$, the inequality follows from the fact that $i = argmin_{k \in F_k} f_k$, and the second inequality follows from the second primal constraint: $x^*_{ij_k} \leq y^*_i$

Summing up for all k (i.e. all the facilities we open) we get:

$$\sum_k f_{i_k} \leq \sum_k \sum_{i \in N(j_k)} y^*_i f_i = \sum_{i \in F'} y^*_i f_i \leq \sum_{i \in F} y^*_i f_i$$

Where the equality is due to the fact that $N(j_k)$ is a partition of F'. ■

Now we are ready to prove the algorithm is 4-approximation.

Proof of theorem 4.2.1. Fix an iteration of the algorithm, and let $i = i_k$ and $j = j_k$. The cost of assigning j to i is $c_{ij} \leq a^*_j$. This follows from the fact that $x^*_{ij} > 0$ and then using complementary slackness and $b_{ij} > 0$. We can bound the cost of assigning an unassigned client $l \in N^2(j)$ to i by:

$$c_{il} \leq c_{ij} + c_{hj} + c_{hl} \leq a^*_j + a^*_j + a^*_l \tag{4.1}$$

where $h \in N(j)$. Since we selected j to be $j = argmin_{j \in temp} a^*_j$, we get that $a_j \leq a_l$, thus $c_{il} \leq 3a^*_l$ So, we get for the assignment cost $\sum x_{ij} c_{ij} \leq 3 \sum_{j \in D} a^*_j \leq 3OPT$ following from weak duality. Finally, using lemma 4.2.2 we get that we are within 4 of OPT.

■

This algorithm contains implicitly a greedy idea in the way it does the rounding. It is natural to assume that when $x_{ij} > 0$ we should probably consider opening this facility and connect j to it, since it is so in the fractional optimal solution, but if we did this blindly for all clients, we could end up paying a very large facility opening cost. This is where the greedy comes in ; we should open the cheapest facility we can, in j's neighborhood, and then use triangle inequality to bound the assignment cost of the N^2 neighbors of i instead of opening a new facility for every client.

4.3 Randomized Rounding

In this section we show a randomized rounding algorithm for the Facility Location problem, which is a modification of the previous deterministic rounding algorithm. In this case, the rounding involves some random variable, hence the name.

In this algorithm, we do not choose deterministically which facility i_k we will open for the client j_k, but we choose the facility according to the distribution x_{ij} instead (note that $\sum_i x_{ij} = 1$). This will improve the previous analysis since, generally, in the deterministic setting we need to make worst case assumptions in order to find upper bounds.

We define $C_j^* = \sum_{i \in F} x_{ij}^* c_{ij}$ - the cost incurred by client j in the fractional LP solution.

Algorithm 3: Randomized rounding for FL

Data: Primal and dual optimal solutions (x^*, y^*), (a^*, b^*)

$temp \leftarrow D$;
$k \leftarrow 0$;
while $temp \neq \emptyset$ **do**
  $k \leftarrow k + 1$;
  $j_k = argmin_{j \in temp}\{a_j^* + C_j^*\}$;
  Choose i_k according to probability distribution x_{ij}^*;
  assign j_k and $\{j \in N^2(j_k) : j \text{ is unassigned}\}$, to i_k;
  $temp \leftarrow temp - \{j_k\} - N^2(j_k)$;
end

Theorem 4.3.1. *Algorithm 3 is a 3-approximation algorithm for Uncapacitated Facility Location*

Proof. As before, fixing an iteration k, the expected facility opening cost is:
$\sum_{i \in N(j_k)} f_i x_{ij_k}^* \leq \sum_{i \in N(j_k)} f_i y_i^*$ (using the second primal constraint).

Since the neighborhoods $N(j_k)$ form a partition of a subset of the facilities, we get:

$$\sum_k \sum_{i \in N(j_k)} f_i y_i^* \leq \sum_{i \in F} f_i y_i^* \tag{4.2}$$

In a certain iteration, let $i = i_k$ and $j = j_k$, the expected cost of assigning j to i is exactly C_j^* we defined previously. Therefore, the expected assignment cost is:

$$c_{ij} \leq c_{hl} + c_{hj} + \sum_{i \in N(j)} c_{ij} x^*_{ij} = c_{hl} + c_{hj} + C^*_j$$

Using again that $c_{ij} \leq a^*_j$ we get that $c_{ij} \leq a^*_j + a^*_l + C^*_j$.

But since $argmin_{j \in temp} a^*_j + C^*_j$ we finally get that the total expected cost is:

$$\sum_{i \in F} f_i y^*_i + \sum_{j \in D} (2a^*_j + C^*_j) = \sum_{i \in F} f_i y^*_i + \sum_{i \in F, j \in D} c_{ij} x^*_{ij} + 2 \sum_{j \in D} a^*_j \leq 3OPT$$

■

In this type of rounding, the randomization inserted by the probability distribution to the algorithm affects only the cost. There are cases, where the randomization affects also the feasibility of the final solution given. In such cases we should use other techniques to ensure that the algorithm will give a feasible solution with high probability.

4.4 The Primal-Dual Approach

In contrast to the two previous methods, in the primal dual method we do not need to actually solve the linear program and then do something with the solution. By exploiting the properties of a dual optimal solution, we construct one without solving the LP, which leads us to a much faster algorithm. This is the technique that will mostly be used in the sections to follow, where we describe the online variant of Facility Location since it has a nice intuitive interpretation for the problem.

This algorithm for Facility Location was proposed by Jain and Vazirani in [12]. It achieves a tight 3-approximation in $m \log m$ running time, where $m = n_c \cdot n_f$ is the number of "edges" between the facilities and the clients ($n_c = |C|$ and $n_f = |F|$).

From primal complementary slackness conditions we get:

- if $i \in F' \Rightarrow \sum_{i:\phi(j)=i} b_{ij} = f_i$
- if j is connected to i, then $b_{i'j} = 0$, $\forall i' \neq i$ and $a_j - b_{ij} = c_{ij}$. We can think of this as a_j being the total price paid by client j, which breaks down to c_{ij}: the contribution of the client towards the edge (i, j) and b_{ij}: the contribution towards facility i

The algorithm works in two phases: in the first phase will find a set of facilities, we name them *temporarily open*-F_t, a set T of tight edges (i, j). Also let Con be the set of connected clients. In the second phase it will choose a set $F' \subseteq F$ of facilities to open, and it will find a mapping $\phi : C \to F'$.

More specifically, in phase 1 the algorithm raises the dual variables a_j. When the client has reached the connection cost, after declaring the edge (i, j) to be tight, he will start to actually contribute to the opening of the facility. This is done by raising also the variable

b_{ij} while maintaining dual feasibility. Since variables a and b increase simultaneously, at some point a facility will be fully paid for i.e. the second dual constraint will be tight, so this facility will be declared temporarily open, and the tight demands will be assigned to it. The algorithm for phase 1 is shown in 4.

Algorithm 4: Jain-Vazirani - Phase 1

```
1  T ← ∅, Con ← ∅, F_t ← ∅;
2  a_j ← 0 ∀j;
3  b_ij ← 0 ∀j, i;
4  while C \ Con ≠ ∅ do
5  |   a_j += 1 ∀j ∈ C \ Con;
6  |   b_ij += 1 for tight edges (i, j);
7  |   if ∃j_1 ∈ C, i_x ∈ F : a_{j_1} = c_{i_x j_1} then
8  |   |   T ← T ∪ (i, j);
9  |   end
10 |   if Σ_j b_ij = f_i then
11 |   |   F_t ← F_t ∪ i;
12 |   |   Con ← Con ∪ j;
13 |   |   φ(j) = i;
14 |   end
15 end
   // The set of special edges
16 Spec = {(i, j) : b_ij > 0};
```

In phase 2, the algorithm needs to decide which facilities from the set of temporarily open will stay open in the final solution, and how will the clients be assigned to them. Phase 2 of the algorithm is shown below in 5. Initially we get $T = G\{Spec\}$ to be the edge-induced subgraph of G using the edges $Spec$. Then we take T^2 which is essentially T with some extra edges when there is a path of length 2 from a node of T to another. Then $H = G[F_t]$ is the vertex-induced subgraph of T using the vertices of F_t. Finally we find a maximal independent set in H and we open these facilities. This means that each client will contribute to at most 1 open facility. Finally, to find the assignment ϕ, in the way we created the graphs, they will either be connected to a facility in the independent set F' (so they will be directly connected) or there will be a facility at most one "hop" from them, so they will be indirectly connected. Essentially we get an idea of where the algorithm would have connected each client, and which facilities whould be open, and then try to

open the best possible subset of facilities to minimize the objective.

Algorithm 5: Jain-Vazirani - Phase 2

```
   // Decide which facilities to open
1  T ← G{Spec};
2  T^2 ← (T, e) where ∃ edge e = (i, j) iff there is a path at most 2 from i to j
     in T;
3  H ← G[F_t];
4  F' ← maximal independent set on H;
5  Open facilities in H;
   // Create assignment φ
6  S_j ← {i ∈ F_t | (i, j) ∈ Spec};
7  if ∃i ∈ S_j : i ∈ I then
8  |   φ(j) ← i;
   |   // j Directly connected
9  end
10 else
11 |   Let tight edge (i', j) : i' is connecting witness of j;
12 |   if i' ∈ I then
13 |   |   φ(j) ← i';
   |   |   // j Directly connected
14 |   end
15 |   else
16 |   |   Let i'' ∈ I be a neighbor of i' in H;
17 |   |   φ(j) ← i'';
   |   |   // j Indirectly connected
18 |   end
19 end
```

The algorithm essentially tries to maintain the primal approximate complementary slackness condition for indirectly connected clients: $1/3c_{\phi(j)j} \leq a_j \leq c_{\phi(j)j}$. Using this, the analysis will be very straightforward. We say that $a_j = a_j^e + a_j^f$ for every dual variable, where the two different type of contributions of client j are the contribution towards opening a facility (a_j^f) and the one towards the connection cost (a_j^e). When j is indirectly connected, he does not contribute towards the opening of any facility, so $a_j^f = 0$ and

$a_j = a_j^e$. If j is directly connected, we have that $a_j = b_{ij} + c_{ij}$ (so $a_j^e = b_{ij}$ and $a_j^f = c_{ij}$).

It is easy to see, that when $i \in F'$ we have that $\sum_{j:\phi(j)=i} a_j^f = f_i$, and since $a_j^f = 0$ for indirectly connected, we have that $\sum_{i \in I} f_i = \sum_{j \in C} a_j^f$.

Using this, and the fact that for indirectly connected clients $c_{ij} \leq 3a_j^e$ it is straightforward to prove that:

$$\sum_{i \in F, j \in C} x_{ij} c_{ij} + 3 \sum_{i \in F} f_i y_i \leq 3 \sum_{j \in C} a_j \leq 3OPT \tag{4.3}$$

Essentially what we do in prima-dual method is that we start with a feasible dual solution and we raise the dual variables, usually in a uniform way until some dual inequalities become tight. This tells us that we should add the corresponding primal variable to the solution. And usually by requiring that the relaxed version of the primal or dual complementary slackness conditions be satisfied, we get the approximation ratio.

Chapter 5

Online Facility Location and Variants

In this section we will present the most important results for the online Facility Location problem and the Sum Radii problem. The combination of these two problems, will give us the new variant of section 6.

In the online Facility Location, we do not know the number and positions of demands, that arrive one by one and need to be irrevocably connected to a facility upon arrival. We will only consider these problems in metric spaces. To the author's knowledge, as for now, the online Facility Location is not studied yet in the non metric setting. There are slight differences in the settings of the prolem, for example in some settings, every demand can open a facility, but in some other settings, the facilities can open anywhere in the metric space. We can think of online Facility Location as a variant of clustering, where not all the points are known beforehand. This is more realistic on the internet for example, where new data are generated continuously and we want to divide them into clusters-teams of similar objects. Another potential motivation for this problem, is if we have a network where every computer needs to be connected to a server by cable, and new users arrive online. We have a cost to purchase the server, and a cost proportional to the length of the cable.

In the online Sum Radii (or Sum Diameters), as before, the demands arrive online, and must either be inside an already open cluster or open a new one. In this case, the objective is to minimize the sum of radii, or the sum of diameters. It is easy to see that these two problems are the same, and the optimal solution of Sum Radii is within 2 of the optimal solution of sum - diameters. For a possible application of this problem, apart from applications to clustering and data analysis, we can think that an internet provider wants to provide internet access via wireless network to people coming to live in an area. The company pays in order to open a new tower, and the larger its radius is, the more power it needs, so it would be more expensive. Every time a new customer arrives, if he

cannot be serviced by an already open tower, the company needs to open a new one, in order for the customer to have wireless internet access.

5.1 Algorithms

5.1.1 Online Facility Location

Meyerson [4] introduced the Online Facility location problem and presented an elegant and very intuitive randomized algorithm, giving $O(\log n)$ against adversarial input and $O(1)$ for demands arriving in random order. Each demand that arrives, must be irrevocably assigned to a facility or open itself one. Note that facilities can only open on points where a demand has arrived.

Let F be the set of currently open facilities, we define $d(u_j, F) = min_{i \in F}\{d(u_j, i)\}$ i.e. the distance of demand u_j to the closest open facility. The algorithm is presented below, when a new point arrives, if δ is its distance from the closest already open facility, we open a facility at this point with probability $\frac{\delta}{f}$.

Algorithm 6: Meyerson OFL

Data: Points set S

while $S \neq \emptyset$ **do**
  take new point p;
  $\delta = d(u_j, F)$;
  with probability $\frac{\delta}{f}$ open facility at p (pay f) ;
  else connect p to closest open facility (pay δ);
end

The intuition behind this is quite simple ; when the demands arrive close to a currently open facility we want them to connect to the facility rather than opening a new one, hence the lower probability. However, when demands arrive further away from currently open facilities, we want them to be able to open a facility, in order to avoid paying a high assignment cost, and potentially cover other far away demands that will arrive in the area later. So, when many demands arrive in a certain area, they will eventually open a facility, having already paid on expectation only f plus f for the opening of the new facility. Note that we cannot change the facility the demands are assigned to, so the demands that contributed to the opening of the new facility will remain assigned to the one they were before, but all subsequent demands in the area will have a closer facility to connect to. The proofs of lemmas 5.1.1 and 5.1.2 were both first presented in [4]

Lemma 5.1.1. *Algorithm 6 is $O(1)$ competitive for random ordered demands*

Proof Sketch. Let C_i^* be the optimal cluster, d_p^* be the distance of point p from closest open facility in the optimal solution, $A_i = \sum_{p \in C_i^*} d_p^*$, $a_i = A_i^*/|C_i^*|$ average distance of points in cluster C_i^* and γ_p the cost point p pays (either as assignment or as facility cost). The half points that are closest to the center are named "good", and the others "bad".

Essentially the "good" points that are closer to the center, will either pay a small amount in connection cost, if there is a facility opened close to the optimal or they will quickly open a facility close to the optimal, so the subsequent ones (that are many) will benefit from it and will not pay much in connection cost. The "bad" ones on the other hand, do not have that nice property, and this may result in them paying to open a facility which will not benefit much subsequent demands.

The proof continues by bounding the expected cost paid by all good points $\mathbb{E}[\sum_g \gamma_g]$ by $2f + 2A_i^* + 2\sum_g d_g^*$ which holds regardless of the order they arrive, and the cost paid by a bad point $\mathbb{E}[\gamma_b]$ by $2d_b^* + \frac{2}{|C_i^*|}(f + \sum_g(\mathbb{E}[\gamma_g] + 2d_g^*))$.

Combining these two inequalities, we get that the expected cost paid by the algorithm, is within 8 of the offline optimal.Note that this holds if the bad points are injected randomly in an ordering of the good points, and are not presented adversarially to the algorithm. ■

When the demands arrive adversarially however, Meyerson [4] showed that no algorithm can be $O(1)$ competitive. However, this analysis is not tight, Fotakis showed [5] that this algorithm is asymptotically optimal i.e. it achieves $\Theta(\frac{\log n}{\log \log n})$ competitive ratio, which is the lower bound for Online Facility Location. We will show proof sketches for both results, for the sake of completeness and since the proof shown in section 6.5 for our variant is similar to Meyerson's proof.

Lemma 5.1.2. *Algorithm 6 is $O(\log n)$ competitive against adversarial input*

Proof Sketch. Let c_i^* be an optimal center. We divide the area around it in zones (cycles with center c_i^*). Zone S_a contains all the demands with $2^{a-1}a_i^* \leq d_p^* \leq 2^a a_i^*$. Note that there are only $\log n$ zones ; if a demand was outside zone $S_{\log n}$, it would pay more than the cost of the cluster.

We will bound the cost paid by each zone: demands in zone S_a will pay at most f until a facility opens at this zone, and all subsequent demands will pay at most $3d_p^*$ to connect to this facility (this results from the triangle inequality). This gives us expected cost of $6d_p^*$.

Finally we need to bound the cost of the points within a_i^*. Since they are the ones closest to the center, they are the ones bound to pay much more than their optimal cost. This is not the case however, since they will also pay an expected f until opening a facility,

and then all subsequent demands will pay at most $2d_p^*$.

Summing up for all zones, we pay $(\log n + 1)f + 8A_i^*$.

■

Note that the $\log n$ factor results essentially from the opening of a facility in each zone, so $\log n$ facilities, and not from the assignment cost.

Lemma 5.1.3. *Algorithm 6 is* $\frac{\log n}{\log \log n}$ *competitive*

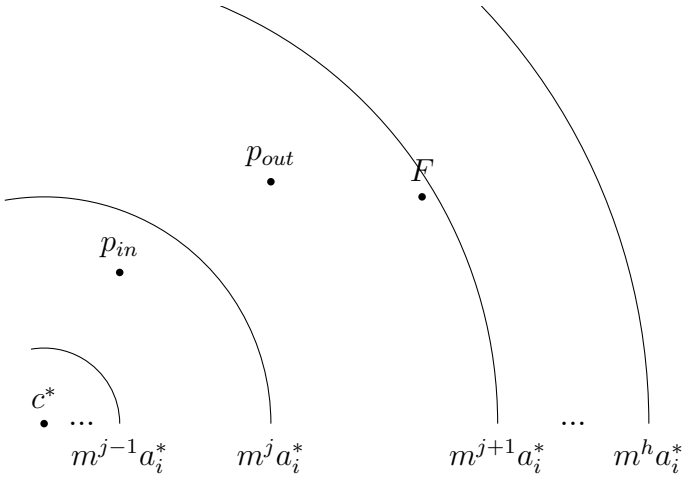

Figure 5.1: Phases for the tight analysis

Proof Sketch. Let m, h be any positive integers such that $m^h > n$ and F^* be the optimal facility cost. In this case, the proof is similar to Meyerson's with the difference that the analysis is divided into h disjoint phases starting from h as seen on figure 5.3. The j'th phase begins when the algorithm opens a facility F such that: $m^j a_i^* \leq d(F, c) \leq m^{j+1} a_i^*$. There is also a phase after phase 0 which never ends. All demands p such that $d_p^* \leq m^j a_i^*$ are inner, and all the others are outer. We will bound the cost of inner and outer demands for a phase separately. This distinction is similar to Meyerson's good and bad demands, ; we will be able to charge the outer demands' cost to their optimal assignment cost (since there is a facility open within distance $m^{j+1}a_i^*$ of the center) but the inner are the ones that can pay much more in our solution, since they pay very little in the optimal.

Outer demands: $d(F, p) \leq d(F, c^*) + d_p^* \leq (m+1)d_p^*$ from the definition of the phases. For the final phase, we get $d(F, p) \leq 2(a_i^* + d_p^*)$. Therefore the total assignment cost for outer demands is $2(m+2)A_i^*$.

Inner demands: since each phase stops when an inner demand opens a facility, we need to bound the cost paid until that facility opens, which is $2f$. Thus, we will pay $2(h+1)F^*$

Therefore, setting $h = m = \frac{\log n}{\log \log n}$ we get a competitive ratio of $\frac{\log n}{\log \log n}$. ■

5.1.2 A Primal Dual Algorithm for Online Facility Location

In [6] Fotakis presented a simple $O(\log n)$ deterministic primal dual algorithm for non uniform Online Facility Location. The setting is the same as Meyerson's Online Facility Location [4]; demands arrive online and we must decide whether they will be assigned to an already open facility or open a new one on a point of the metric space. The assignment and opening decisions are irrevocable. Although, note that in this setting, facilities can open on any point of the metric space, while in Meyerson's setting, facilities only open on points of demands.

Let (M, d) be a metric space, L the set of demands seen so far, and F the current facility configuration. For each point there is a facility opening cost f_z. For each point $z \in M$, we define its potential as $p(z) = \sum_{j \in L} \max\{(d(F, j) - d(z, j), 0\}$. The LP relaxation for the problem is shown in tables 5.1 and 5.2 ; variables x_{zj} indicate whether demand j is connected to facility z, y_z indicate whether facility z is open, and dual variables a_j show essentially how much each client pays towards the solution.

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{z \in M} y_z f_z + \sum_{z \in M, j \in L} x_{zj} d(z, j) & \\
\text{subject to} & \displaystyle\sum_{z \in M} x_{zj} \geq 1 & , \forall u_j \\
& x_{zj} \leq y_z & , \forall z \in M, j \in L \\
& x_{zj} \geq 0,\ y_z \geq 0 & , \forall (z, j)
\end{array}
$$

Table 5.1: Primal relaxation for Online Facility Location

The algorithm maintains the invariant that the potential for every point in M is less than the cost of opening a facility at this point. If we set the dual variables a_j to be the distance of the demand from the currently open facility, this invariant is essentially ensuring dual feasibility. So when the constraint is violated, we open a facility at the point of the most violated constraint. This is where linear programming and duality come in ; the notion of the "potential" of a point is a quite natural thought and we see that it also appears as a result of a mechanical process - when deriving the dual from the primal.

$$\begin{array}{lll} \text{maximize} & \sum_{j \in L} a_j & \\ \text{subject to} & \sum_{j \in L} \max\{a_j - d(z,j), 0\} \leq f_z & ,\forall z \in M \\ & a_j \geq 0 & ,\forall u_j \end{array}$$

Table 5.2: Dual program for Online Facility Location

The intuition behind the invariant is similar to the one for Meyerson's probability in the randomized Online Facility Location. When a point has high potential, it means that many demands will benefit from opening a facility there, and they will pay towards it (through variable a_j). The potential aims to quantify exactly this amount: how much do nearby demands want a demand in point $z \in M$. In a sense, this says that the facility locations converge to the optimal positions, since every demand contributes positively only when the potential new facility is closer to it than the one it is assigned to. This means, that many demands will cause a facility to open closer to them. The algorithm is shown in 7.

After showing that the invariant is maintained by the algorithm in each step, Fotakis bounds the algorithm's assignment cost by $\log(n+1)F^* + (2(\log n+1)+1)Asg^*$ and the algorithm's facility cost by $\sum_{j \in L} a_j$ which is in turn bounded by $(\log n+1)F^* + (2\log n+1)Asg^*$, thus giving us the $O(\log n)$ competitive ratio. It is still an open problem to find a tight example for this algorithm or improve the analysis to give a better than the logarithmic competitive ratio.

5.1.3 Simple Deterministic Online Facility Location

In [7] Anagnostopoulos et al. presented a simple deterministic algorithm for uniform online Facility Location, based on hierarchical partition of the plane, which gives $O(\log n)$ competitive ratio. This algorithm works for both Meyerson's setting, where facilities are only placed on demands, and Fotakis' setting where facilities can be placed anywhere on the plane.

The algorithm's idea is quite simple, and uses -in a way- the concept of the "potential" of an area, used explicitly in [3] and [6], and implicitly in Meyerson's randomized algorithm [4]. The algorithm starts with a quadrant of diagonal length f - an assumption which is generalized later - which opens a facility and is partitioned into 4 smaller ones (figure 5.2), once the cost of the demands associated with this exceeds a threshold.

Algorithm 7: Fotakis' Online Facility Location Primal-Dual

Data: Points set S

$F \leftarrow \emptyset$, $L \leftarrow \emptyset$, $p(i) = 0,\ \forall i \in M$;
while $S \neq \emptyset$ **do**
  take new point $p \in S$;
  $L \leftarrow L \cup \{p\}$;
  foreach $z \in M$ **do**
    `// Update potentials`
    $p(z) \leftarrow p(z) + \max\{d(F,p) - d(z,p), 0\}$;
  end
  `// Find most violated`
  $w \leftarrow argmax_{z \in M}\{p(z) - f(z)\}$;
  if $p(w) > f_w$ **then**
    $F \leftarrow F \cup \{w\}$;
    `// Calculate new potentials`
    foreach $z \in M$ **do**
      $p(z) \leftarrow \sum_{u \in L} \max\{d(F,u) - d(z,u), 0\}$
    end
  end
  assign p to nearest open facility;
end

More specifically, we say that the demands that arrive in a quadrant are its *support*, and the total assignment cost of a quadrant Q (*cost(Q)*) is the assignment cost of all the support demands. We also say that the quadrant Q is *open* if it has an open facility associated with it, else it is *recruiting*. We define $d(p,F) = \min_{f_i \in F} d(p,f_i)$ to be the minimum distance of point p from the set of open facilities F.

It is easy to see that the algorithm cannot partition the plane into more than $\log n$ levels ; in every quadrant of depth i the distance of a demand in this quadrant to the closest facility is at most $f/2^i$ (the distance to the facility of the parent quadrant, that was partitioned in a previous step). In order for this quadrant to be partitioned, it needs to have potential (cost of support demands) more than $a2^i$. Therefore if all the n demands arrive always in the same quadrant, we can have at most $\log n - \log a$ levels.

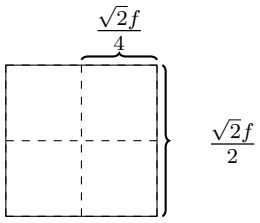

Figure 5.2: Breaking of initial square into quadrants

Algorithm 8: Simple DFL

Data: Points set S

while $S \neq \emptyset$ **do**
  take new point p;
  $Q \leftarrow$ quadrant of p;
  $support(Q) \leftarrow support(Q) \cup p$;
  $cost(Q) \leftarrow cost(Q) + d(p, F)$;
  if cost$(Q) > af$ **then**
    $partition(Q)$;
    open facility at center of Q;
  end
  assign closest p to closest f_i;
end

In the proof, the authors proceed to show that for each optimal facility c^* there is a constant number of quadrants of a certain depth, which contain demands allocated to c^*. In turn, the cost of these quadrants is again constant. This means, that summing up for all levels, for each optimal facility we get a $\log n$ factor approximation. The key observation that allows us to get the constant number of quadrants and constant cost is that in the optimal solution, there is a facility not too far away from each quadrant.

Intuitively, we can think that we have an image that is revealed to us in steps. When we partition a quadrant, it is like increasing the resolution of the image in this part ; since there is much information there, we need more resolution to distinguish the details. The threshold quantifies the amount of information that is enough to need better image quality. The difference between this algorithm and the Primal-Dual one, that both use the notion of potential, is that in this case the decomposition of the metric space is more restricted in a way ; we partition a certain quadrant into certain parts and consider the potential only in this area, while in the Primal - Dual one, the potential can cause a facility to open anywhere.

Algorithm 9: The *partition* function

```
Data: Quadrant Q
X ← break Q into 4;
foreach new quadrant q do
    support(q) ← ∅;
    cost(q) ← 0;
end
Quadrants ← (Quadrants − {Q}) ∪ X;
```

The restriction that the area should be enclosed in a square of diagonal length f is easily dropped. If the region is larger, we can "cover" it with more than one square of diagonal length f, and execute the algorithm in each one separately, depending on where each new client arrives.

In Meyerson's model, the only modification the algorithm needs, is that we will open the new facility in the demand closest to the center of the quadrant instead of the center itself. And since in the proof, the authors used only the size of the quadrant and not the location of the facility, the algorithm is $\log n$ competitive in this setting too.

5.1.4 An Optimal Deterministic Algorithm

Fotakis in [3] presented an optimal deterministic algorithm, that even though it achieves the $\frac{\log n}{\log \log n}$ approximation ratio, it is not practical in use, and has a complicated analysis. We will briefly present the algorithm here, without getting into the details of the analysis.

The algorithm uses the set of unsatisfied demands L, to ensure that each demand will contribute at most once to the cost of the algorithm.

The algorithm opens a facility only when the potential of a certain Ball around the newly arrived demand exceeds the facility opening cost f, where $\text{Ball}(u, r) = \{u \in M : d(u, v) \leq r\}$. This idea seems familiar, since all the previous algorithms used it to some extend. The two main differences in this case are the area which we consider in order to open a facility ($\text{Ball}(p, r_p)$) and the location of the new facility to open, which is chosen more carefully. Specifically, when the condition for opening a new facility holds, we try to find the smaller ball inside B_p such that: the potential accumulated is more than half the potential of the large ball or for every smaller ball the potential is less than half. When the facility is opened, all the demands inside B_p become satisfied, so they cannot contribute in the opening of a facility later on.

The main result of this work is the following theorem.

Theorem 5.1.4. *For $x \geq 10$ the competitive ratio of algorithm 10 is $\frac{\log n}{\log \log n}$*

Algorithm 10: Optimal Deterministic OFL

Data: Points set S

$x \geq 10$;
$F \leftarrow \emptyset, L \leftarrow \emptyset$;
while $S \neq \emptyset$ **do**
  take new point p;
  $L \leftarrow L \cup \{p\}$;
  $r_p \leftarrow d(F,p)/x$;
  $B_p \leftarrow \text{Ball}(p, r_p) \cap L$;
  $Pot(B_p) \leftarrow \sum_{u \in B_p} d(F,u)$;
  if $Pot(B_p) \geq f$ **then**
    if $d(F,w) \leq f$ **then**
      Find the largest v such that:
       $Pot(B_p \cap \text{Ball}(\hat{p}, r_p/2^v)) > Pot(B_p)/2$ and
       $\forall u \in B_p, Pot(B_p \cap \text{Ball}(u, r_p/2^{v+1})) \leq Pot(B_p)/2$;
    end
    else
      $\hat{p} \leftarrow p$;
    end
    $F \leftarrow F \cup \{\hat{p}\}$;
    $L \leftarrow L \setminus B_p$;
  end
  assign p to nearest facility in F;
end

In the analysis presented in [3], Fotakis proceeds to bound the cost of one optimal center, by dividing the area into disjoint phases according to the distance of the optimal center and the algorithm's facility configuration, and then using a non trivial potential function argument. Then, dividing the optimal centers into groups -or *coalitions*- of centers that are closer to each other than any of the algorithm's facility, he bounds the cost for arbitrary many optimal centers. It is also proved that this algorithm can be generalized to give the $\frac{\log n}{\log \log n}$ approximation for non-uniform facility location also.

All these algorithms for online Facility Location use the same implicit idea, which is maybe more apparent in Anagnostopoulos' et al. algorithm of section 5.1.3 ; they all try to approach the optimal position of the facility using binary search. In Anagnostopoulos' algorithm this is quite clear since we begin with a large quadrant, saying initially that the

optimal is somewhere inside this, but using the potential of the sub areas, we divide each time into 4 parts when we think we found the location with better accuracy. The potential in each of the above algorithms, is essentially what guides us in this binary search, and tells us how close we really are to the optimal position.

5.1.5 Online Sum Radii

Fotakis and Koutris in [19] presented one deterministic primal - dual and one simple randomized algorithm for the Sum - Radii problem. In this problem, demands arrive online and we want them to be covered by a facility, either by connecting them to an already open one (if they lie inside its radius) or by opening a new one. The objective in this case is to minimize the sum of radii and opening costs for each open facility.

Let (M, d) be a general metric space, and $N = \mathbb{N} \cup \{-1\}$. For simplicity, we can say that radii are $r_k = 2^k f$ by losing a factor 2 in the approximation as we will show in proposition 5.1.5. The LP-relaxation of the problem is shown in table 5.3

minimize	$\sum_{(z,k)\in M\times N} x_{zk}(f + r_k)$			maximize	$\sum_{j=1}^{n} a_j$		
subject to	$\sum_{(z,k):d(u_j,z)\le r_k} x_{zk} \ge 1$	$\forall u_j$		subject to	$\sum_{j:d(u_j,z\le r_k)} a_j \le f + r_k$	$\forall (z,k)$	
	$x_{zk} \ge 0$	$\forall (z,k)$			$a_j \ge 0$	$\forall u_j$	

Table 5.3: Primal - Dual for the Sum-Radii problem

Lemma 5.1.5. *Let $C(z, r)$ be the clusters that the optimal solution to Sum-Radii uses. We can use radii $2^i f$ and lose a factor 2 from the OPT solution.*

Proof. Let $k = \max\{\log(r/f), 0\}$. For each cluster $C(z, r)$ of the optimal solution, we will open one of radius $2^k f$. The optimal cluster pays $f + r$ while we pay $f + 2^k f$. If $r < f \Rightarrow k = 0$ we pay $2f$. If $f \le r$ we pay $f + 2^{\log r/f} f = 2r$. So in every case, we are within 2 of the optimal. And since we open a cluster of radius r of $f > r$ our solution will be feasible. ■

In the integer version of the problem, variables $x_{zk} \in \{0, 1\}$ indicate whether there is a cluster of radius r_k open on point z of the metric space, whereas in the above relaxation they indicate the extend to which the cluster is open. In the primal constraint, we require that every demand is covered. In the dual problem, each variable a_j corresponds to the "amount" each demand u_j pays towards the solution, while we require that no facility is overpaid.

The primal - dual algorithm is shown in 11 ; it maintains a dual feasible solution by updating the variables a_j for each new demand that arrives, opening a suitable cluster when a dual constraint becomes tight.

More specifically, when a demand arrives, if it is already covered by a cluster $C(z, r_k)$, we assign it to z and set its dual variable to 0, since it did not contribute anything to the opening of a facility.

On the other hand, if the demand is not covered, its dual variable becomes f so that at least the constraint of the $(u_j, -1)$ cluster is tight, in order to maintain feasibility: the demand will not be left uncovered. If more than one constraints are tight, we open a cluster on the point $z \in M$ that we obtain the maximum radius. Intuitively we open the largest cluster that is already paid for by the demands.

Algorithm 11: FL_Radii_Primal_Dual

Data: F: Set of open clusters

$F \leftarrow \emptyset$;
foreach *new demand* u_j **do**
  if u_j *is covered by* c_i **then**
    assign u_j to c_i;
    $a_j \leftarrow 0$;
  end
  else
    $a_j \leftarrow f$;
    $x \in M : \sum_{j:d(u_j,z\leq r_k)} a_j = f + r_k$ and $\nexists k' > k : \sum_{j:d(u_j,z\leq r_{k'})} a_j = f + r_{k'}$;
    $F \leftarrow F \cup \{C(x, 3r_k)\}$;
  end
end

Dual feasibility follows easily from the fact that variables a_j are either 0 or f, so the dual constraint cannot be violated before becoming tight. Additionally, every time we open a new cluster, all subsequent demands that could increase the left-hand side of the tight constraint will be covered, so they will have $a_j = 0$.

Lemma 5.1.6. *Algorithm FL_ Radii_ Primal_ Dual is* $O(\log n)$ *competitive.*

Proof Sketch. Initially we observe that clusters $C(z, k)$ where $k \geq \log n$ cannot exist ; the constraint will never be tight since the right hand side will be nf but the left hand side is always $< nf$.
For $k \leq \log n$ it is easy to see that every demand u_j with $a_j > 0$ will potentially cause at most one cluster $C(z, 3r_k)$ to open, by making $C(z, r_k)$ tight (we reach a contradiction if we assume that u_j caused more clusters to become tight). Therefore since we open clusters

$C(z, 3r_k)$ when $C(z, r_k)$ is tight, the cost for each cluster is at most $cost(C(z, 3r_k)) = \sum_{u_j \in C(z,r_k)} 3a_j$ so the total cost of our solution is $\sum_{(z,r_k):C(z,r_k)\ opens} cost((C(z, 3r_k))) \leq \sum_{(z,r_k):C(z,r_k)\ opens} \sum_{u_j \in C(z,r_k)} 3a_j \leq \log n \sum a_j$

■

The randomized version of this algorithm also gives a $\log n$ approximation. In this case, for the uncovered demands the algorithm tries to open clusters of exponentially large radii with geometrically reduced probability i.e. radii $2^i f$ with probability $1/2^i$. The intuition behind this is that we want to open some large clusters, in order to cover more demands, but not too many, for we will pay too much.

Algorithm 12: FL_Radii_Rand

```
New demand u_j;
if u_j is covered by c_i then
    assign u_j to c_i;
end
else
    With p_uncov = 1/2^i, ∀i ∈ [log n] open facility at u_j with radius 2^i
end
```

This algorithm will be one of the building blocks of the algorithm presented in the next chapter, for our new variant. The proof of the following lemma given in [19] will be presented in detail since we will use the main idea in the proof of the competitive ratio in section 6.5.

Lemma 5.1.7. *Algorithm FL_Radii_Rand is $O(\log n)$ competitive.*

Proof. We bound the expected cost of the algorithm until they open a facility of radius 2^{k+1} which will entirely cover c_k.

Let $T_1 \in \mathbb{N}$ be the time when the cluster of size 2^{k+1} opens and let X_i be the random variable of the cost paid by the algorithm for each new demand u_j. The total cost paid by the algorithm until time T_x is $\sum_{i=1}^{T_x} \mathbb{E}[X_i]$. However, we cannot know the probability of a demand arriving covered (thus paying 0) and arriving uncovered (thus paying the opening and radius cost). In order to bound the cost, we define a random variable Y_i which is the cost paid for each demand as if they al arrive uncovered, therefore, following from this definition: $X_i \leq Y_i$ hence

$$\sum_{i=1}^{T_x} \mathbb{E}[X_i] \leq \sum_{i=1}^{T_y} \mathbb{E}[Y_i] \tag{5.1}$$

Note that the stopping times are different for variables Y_i and X_i ($T_y < T_x$), since X_i do not always try to open a 2^{k+1} cluster while for Y_i in each round there is a non zero probability of opening one. This is not a problem for 5.1 since the left-hand side sum is not increased for the times X_i does not try to open a large cluster.

Recall from algorithm 7 that for each demand we try $i \in [\log n]$ times with probability $\frac{1}{2^i}$ to open facility of radius 2^i. Therefore:

$$\mathbb{E}[Y_i] = \sum_{i=0}^{\log n} \frac{1}{2^i}(f+f2^i) = \sum_{i=0}^{\log n} f+\frac{f}{2^i} = f(\log n+2)+f(2-2^{-(\log n+1)}) \leq f(\log n+2)+2f$$

Clearly T_y is a stopping time since $T_y \in \mathbb{N}$ and depends only on the previous values of Y_i. Also $T_y \sim \mathrm{G}_0\left(2^{-(k+1)}\right)$ thus $\mathbb{E}[T_y] = 2^{k+1}$

The expected cost until time T_y is $\mathbb{E}[cost] = \mathbb{E}[\sum_{i=0}^{T_y} Y_i]$. We observe that Y_is are iid with the same mean ($\mathbb{E}[Y]$) and T_y has finite expectation, so we can use Wald's equation to get:

$$\mathbb{E}[cost] = \mathbb{E}[Y]\mathbb{E}[T_y] \leq 2^{k+1}(f(\log n + 2) + 2f)$$

Adding the cost for the large cluster, the total cost is:

$$cost \leq 2^{k+1}(f(\log n + 2) + 2f + f) + f \tag{5.2}$$

■

5.1.6 Incremental k - Sum Diameters

In [18] Charikar and Panigrahy also presented a constant bicriteria approximation algorithm for Sum k diameters in the incremental setting. In this problem, we need to cover points in the plane that arrive online, by opening at most k clusters/facilities with a certain radius. The objective is still to minimize the sum of radii (or equivalently the sum of cluster diameters). The main difference that distinguishes this problem from a pure online is that the choice of cluster center and radius is not irrevocable ; we can change the radius of an open cluster and possibly close an opened cluster. Charikar and Panigrahy presented an algorithm that uses at most $4k$ centers with cost within 160 of optimal.

The LP formulation of the problem is shown below. For the variables we have that: $y_i^{(r)} = 1$ iff there is an open center at point i with radius r and each dual variable a_j corresponds to a demand point.

minimize	$\sum_{i,r} y_i^{(r)}(r+f_i)$	maximize	$\sum_j a_j$
subject to	$\sum_{i,r:d(i,j)\leq r} y_i^{(r)} \geq 1 \quad ,\forall j$	subject to	$\sum_{j:d(i,j)\leq r} a_j \leq r+f_i \quad ,\forall i,r$
	$y_i^{(r)} \geq 0 \quad ,\forall i,r$		$a_j \geq 0 \quad ,\forall j$

Table 5.4: LP formulation of the problem, $i \in F$ and $j \in C$

We define two sets that are used in the algorithm.

$\mathcal{C}$: core clusters, set of currently open clusters (i.e. the solution to the problem for the points seen so far)

W: witness set, all the points whose dual variable is $a_p = \frac{L}{k}$ in the current phase. Additionally, a cluster $C_i(r)$ is called *near tight* if $\sum_{j:j\in C_i(r)} a_j \geq r/2 + L/k$.

Algorithm 13: Charikar - Panigrahy SinglePhaseSumDiam(S, L)

```
Data: Points set S, Lower Bound L
1  while S ≠ ∅ and |W| ≤ 4k do
2  |  Take new point p;
3  |  if p ∈ near tight cluster then
4  |  |  goto 1;
5  |  end
6  |  else
7  |  |  W ← W ∪ {p}, a_p = L/k;
8  |  |  IncrMerge();
9  |  end
10 end
11 SinglePhaseSumDiam(C ∪ S, 2L);
```

The algorithm tries to maintain a feasible dual solution with at most $4k$ points, which automatically gives a lower bound on the primal's value. The *SinglePhaseSumDiam* represents a single phase of the algorithm, which sets a lower bound for the algorithm that translates to an upper bound for the radii values that the algorithm uses to try to cover the points. Essentially in each phase the algorithm sets an upper bound for the radii that the open centers can have, and tries to cover everything with at most this radius and at most $4k$ centers. If this fails (which means that the $4k$ centers are too few and the upper

bound on the radius too restrictive), in the next phase the radii can be larger, in order to cover more area with less centers.

The witness set is essentially the points that "contribute" to the opening of a center or the points where the center is opened. In each phase, there are some points that will not be processed again later in the algorithm. Those points are of two types:

- Were covered by an open cluster on arrival (so $a_p = 0$)
  In this case, they will always be covered either by the same cluster $C_i(r)$ (which will be passed on to the next phase through the set $\mathcal{C}$ and can only increase its radius) or by another cluster $C_j(r')$ which will fully cover $C_i(r)$. However they will never contribute anything to "enlarging" the radius of the cluster about to cover them.

- Were not covered on arrival, but were at some point part of the witness set ($a_p = L_z/k$) of some phase z of the algorithm
  In this case, at some point they intersected with another opened cluster, which covered them and was removed from the solution. However, since $a_p \neq 0$ they will still "help" the cluster covering them to obtain larger radius.

The function that merges the clusters:

Algorithm 14: IncrMerge()

Data: Core set $\mathcal{C}$, *Sol* set of clusters in current solution
Examine new near tight clusters in *decreasing* order of radius;
Let $C_i(r)$ the currently examined cluster;
if $C_i(r) \cap \mathcal{C} = \emptyset$ **then**
    $\mathcal{C} \leftarrow C \cup C_i(r)$;
    $Sol \leftarrow Sol \cup C_i(5r)$;
end
else
    foreach $C_{i'}(r')$ *that* $C_i(r)$ *intersects* **do**
        if $r' \geq \frac{r}{2}$ **then**
            // $C_i(r)$ is already covered by $C_{i'}(r') \in Sol$
            break;
        end
        else
            // $C_i(5r)$ covers all sets intersecting with $r' < \frac{r}{2}$
            $\mathcal{C} \leftarrow (\mathcal{C} \cup C_i(r)) \setminus C_{i'}(r')$;
            $Sol \leftarrow (Sol \cup C_i(5r)) \setminus C_{i'}(5r')$;
        end
    end
end

This is a simple merge rule: just try not to have overlaps in the core set i.e. "spread out" the centers as far as possible, to cover more area. Note that clusters $C_i(r)$ do not have overlaps, but the clusters $C_i(5r)$ may do.

In the last two algorithms for online sum-diameters, the implicit idea we mentioned for the online facility location algorithms is also present; both algorithms use implicitly binary search in order to approach the optimal radius, and not the optimal position this time. This may be more apparent in Charikar and Panigrahy's algorithm than Fotakis and Koutris' randomized one, since each time we double the largest possible radius a cluster can have and try again to cover the area. In the next chapter we will see how to combine the algorithms for the two different problems into one.

5.2 Lower Bounds

In this section, we will briefly present some important lower bounds for the above variants. Specifically, we will prove the $\frac{\log n}{\log \log n}$ lower bound for Online Facility location presented in [3], and briefly present the one for Sum Radii presented in [19]

5.2.1 Online Facility Location

Theorem 5.2.1 (OFL Lower Bound)**.** *Every randomized algorithm for Online Facility Location cannot be better than* $\Omega(\frac{\log n}{\log \log n})$*-competitive against an oblivious adversary.*

Proof. We will prove this claim using Yao's Lemma, which was described in section 3.3.2, by defining a probability distribution on the data, and proving that every deterministic algorithm cannot perform better than $\frac{\log n}{\log \log n}$. We will construct the lower bound on a binary HST of height h, the construction of the lower bound can be seen in figure 5.3.

Let OPT be the cost paid by the optimal and ALG the cost paid by any deterministic algorithm. We will show that OPT's cost is at most $f + hD/(m-1)$ and ALG's cost is at least $\min\{f/2, Dm\}$ for the first $(h-1)$ phases, and $\min\{f, Dm\}$ for the last phase. Using these, and setting $m = h$ and $D = m/h$ we get

$$\frac{ALG}{OPT} \geq \frac{(h+1)f/2}{(2h+1)f/(h-1)} = \frac{(h+1)(h-1)}{2(2h+1)} \approx h \tag{5.3}$$

and since the total demands are m^h, we need $m^h \leq n$, so setting $h = \frac{\log n}{\log \log n}$ we get the desired result.

Specifically, the demands arrive in the following manner: at level 0 there will be only 1 demand, and on every level i after the root, lets say we look at node u_i, we will choose a

node from u_i's children uniformly at random, and place m^i demands there. The optimal solution will open a facility at level h, thus incurring cost at most: $f + hD\frac{m}{m-1}$ which is the cost of one facility and m^i demands paying assignment cost in each level $0 \leq i < h$.

As for the cost of the algorithm, we will find a lower bound in each phase, for the demands in the subtree we will not "see again". So, for example, if the distribution has chosen the right child of u_i, we will bound the cost for the demands arriving on the left child's subtree. Specifically, we fix the adversary's choices until u_i. We distinguish the cases below:

ALG has not opened a facility in T_{u_i}, the demands in $T_{u_i} \setminus T_{u_{i+1}}$ will either open a facility (paying f) or pay assignment cost $m^i D/m^{i-1} = Dm$, so in every case: $\min\{f, Dm\}$.

ALG has opened a facility in T_{u_i}, then with probability $1/2$ there will be a facility in $T_{u_i} \setminus T_{u_{i+1}}$, so demands will pay $\min\{f/2, Dm\}$.

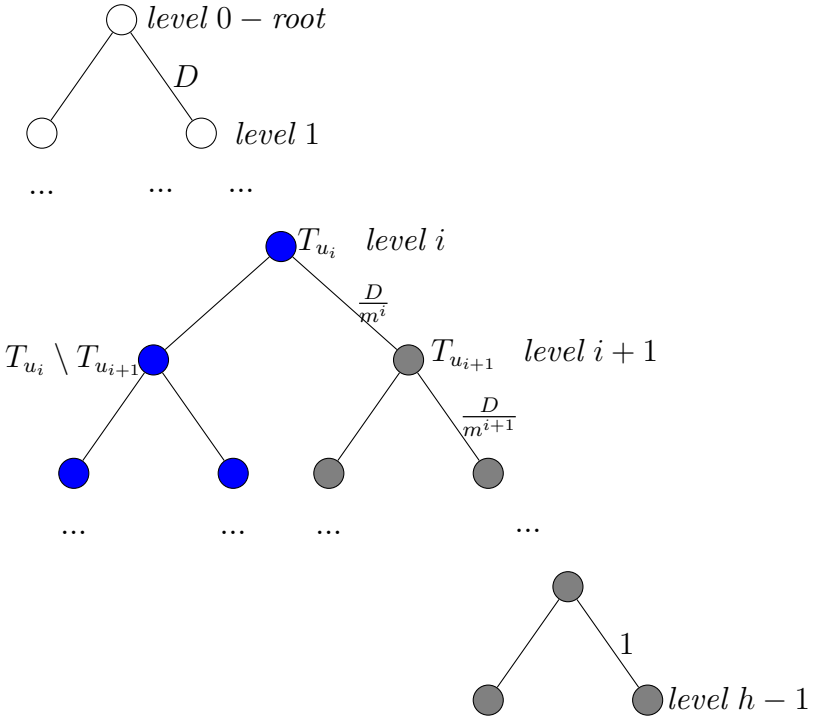

Figure 5.3: The HST used for the lower bound, gray nodes are in the $T_{u_{i+1}}$ subtree, blue are in $T_{u_i} \setminus T_{u_{i+1}}$

■

What this lower bound says essentially, is that with this distribution, in the HST we

will either pay the lower bound as assignment or as facility cost.

5.2.2 Sum Radii

For the Sum Radii problem, Fotakis and Koutris in [19], initially present a $\Omega(\log n)$ lower bound for deterministic algorithms for Sum - Radii on tree metrics, using ternary HSTs. Then they proceed to prove a $\Omega(\log n)$ lower bound, again for deterministic algorithms on the Euclidean plane. While this proves that the deterministic algorithm presented later in the paper is tight, the $O(\log \log n)$ lower bound for randomized algorithms on the problem leaves it open to improve the current $\log n$-competitive randomized algorithm.

Chapter 6

Radii Facility Location

In this thesis, we introduce a new variant which is essentially a combination of the classic online Facility Location and the online Sum Radii problem.

In this variant, each facility opens with a radius fixed at opening time, and it can only service demands within this radius. Each facility pays an opening cost, and a radius cost, while each client pays its distance to the facility it is assigned. This is an online problem, so the demands are not known from the beginning, and once a facility opens, it cannot close or change its radius. Initially we formulate this as an integer program. The IP and its relaxation are presented in detail in the following sections. We will see that we need to scale the assignment cost for the problem to be meaningful, and we provided a randomized algorithm depending on this scale factor. Finally, we studied how the competitive ratio changes for all values of the scale factor.

One possible motivation for this problem is to think that clients arrive in an area and want to have access both to wifi and to cable, so they need, upon arrival, to be inside the radius of a facility. In the end, we will pay the opening cost of each facility, a radius cost, since the larger the radius the more power we need to operate it, and a connection cost, so that each client is also connected with a cable to the facility that covers it.

6.1 LP Formulation

Let F be the set of facilities, C the set of Clients/Demands, f : the initial opening cost, $j \in C, i \in F, r \in \mathbb{R}$. For the radii cost, we we will use clusters of size $2^i f$, as in the Sum Radii problem, and loose at most 2 in the approximation factor. This was proved for Sum Radii in lemma 5.1.5, which is easy to see that applies also to our problem without any changes in the proof.

Additionally, in order for the problem to have meaningful cases, we scale the assignment cost with f_3. We will see in section 6.4 that if we did not have this scale, i.e. $f_3 = 1$, the

problem would have a trivial and optimal algorithm.

In the integer version, $y_i^{(r)}$ indicates if facility i is open with radius r, and $x_{ij}^{(r)}$ indicates if demand j is connected to facility i within radius r.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i,r} y_i^{(r)}(f + fr) + \sum_{i,j,r} x_{ij}^{(r)} f_3 d(i,j) & \\
\text{subject to} \quad & x_{ij}^{(r)} \leq y_i^{(r)} & ,\forall i,j,r \\
& \sum_{i,r} x_{ij}^{(r)} \geq 1 & ,\forall j \\
& x_{ij}^{(r)} \geq 0 & ,\forall i,j,r \\
& y_i^{(r)} \geq 0 & ,\forall i,r
\end{aligned}
$$

Table 6.1: Primal formulation of the problem, $i \in F$ and $j \in C$

In the above primal, we want to minimize the sum of the radii cost, the facility opening costs and the sum of the assignment costs for each client. The first condition says that a client can be connected to a facility only if it is opened. The second one says that every client j should be connected to at least one facility (in the fractional solution a client can have non zero connection to more than one facility).

$$
\begin{aligned}
\text{maximize} \quad & \sum_{j \in C} b_j & \\
\text{subject to} \quad & b_j \leq a_{ij}^{(r)} + f_3 d(i,j) & ,\forall j,i,r : d(i,j) \leq r \\
& \sum_{j \in C} a_{ij}^{(r)} \leq f + fr & ,\forall i,r \\
& a_{ij}^{(r)} \geq 0 & ,\forall i,j,r \\
& b_j \geq 0 & ,\forall j
\end{aligned}
$$

Table 6.2: Dual formulation of the problem, $i \in F$ and $j \in C$

The intuition behind the dual variables is that b_j is what client j pays for its part of the solution and $a_{ij}^{(r)}$ is the share of client j for facility i. In the above dual, we want

to maximize what all clients pay. The first constraint says that the cost each client pays should not exceed it's share for a certain facility plus it's assignment cost. The second one, says that no facility can be "overpaid" (we should not pay for a facility more than it actually costs).

Note that this is a typical covering-packing primal dual since all coefficients $a_{ij}^{(r)}, d(i,j), f+fr$ are non negative.

6.2 Complementary Slackness Conditions

In this section we briefly present the primal and dual complementary slackness conditions with the intuitive meaning of each one. These more or less are the same for these types of primal dual problems.

- $a_{ij}^{(r)} > 0 \Rightarrow x_{ij}^{(r)} = y_i^{(r)}$: If client j has paid for facility i with radius r then either j is connected to i, or the facility i is not open (and client is not connected to it
- $b_i > 0 \Rightarrow \sum_{i,r} x_{ij}^{(r)} = 1$: every client who has paid a non zero cost towards the solution, will be connected to a facility (contrapositive: $\sum_{i,r} x_{ij}^{(r)} \neq 1 \Rightarrow b_j = 0$)
- $y_i^{(r)} > 0 \Rightarrow \sum_j a_{ij}^{(r)} = f + fr$: Every open facility is fully paid for
- $x_{ij}^{(r)} > 0 \Rightarrow b_j = a_{ij}^{(r)} + d(i,j)$: If j is connected to i with radius r the cost he pays is exactly the assignment cost $d(i,j)$ plus the cost towards opening the certain facility

6.3 The Scale Factor

Let for simplicity $f_3 = 1/2^M$. We will see that for different values of M the problem changes from Facility Location (when $f_3 > 1$) to Sum Radii, when $f_3 < 1/2^n$ as seen in figure 6.1. More specifically, we distinguish the cases below:

- $f_3 > 1$: In this case, OPT will never open clusters with radius larger than f since is will always be cheaper to open a facility on the demand than connecting it to an open facility.
- $1 \geq \frac{1}{2^M} \geq \frac{1}{2^{\log n}}$: In this case, OPT will have profit when opening larger clusters than f, but how much larger? It is easy to see, that OPT will never open a cluster larger than 2^{M+1}, for the same reason as above, it will be more expensive to connect a demand this far away rather than opening a facility.

- $\frac{1}{n} \geq f_3$: OPT will never open a cluster larger than n, even if the assignment cost is multiplied by $1/n$

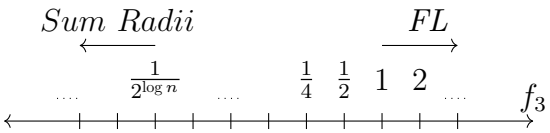

Figure 6.1: Chance of the problem for different values of f_3

So, in every case, OPT may open a cluster at most $min\{2^M, 2^{\log n}\}$, therefore we only need to try to open radii of at most $min\{M, \log n\}$. This is a result of the fact that since the radius cost is scaled as multiples of the facility cost, and the assignment cost has as a natural upper bound the radius, the demand can have as much cost as paying for a facility of δ to open. This fact, led us to scale down the assignment cost, in order for it to not "count" as much in the solution, to allow the OPT to open larger clusters.

6.4 The Algorithm

The algorithm, shown below, is a combination of Meyerson's randomized algorithm, in the case the demand is covered, and Fotakis' and Koutris' randomized algorithm for Sum Radii-in the case the demand arrived uncovered. In the case of a covered demand, we open a cluster with half the radius of the cluster that covers the demand, including in the probability the radius cost. In the case of an uncovered demand, our only modification is that the larger cluster we try to open - with the accordingly changed probability- is not nf, but the larger one that could be opened by optimal, depending on f_3. In the algorithm below, we have facilities c_i with radii $r_i = 2^{u_i}$, and clients/demands p.

Algorithm 15: FL_Radii

Data: M

New demand p:
if p *is covered by* c_i **then**
  $\delta = min\{d(i,p)|i \in F\}$;
  `//` 2^{u_i} `rad of facility covering` p
  $u = u_i - 1$;
  With $p_{cov} = \frac{\delta}{2^u f + f}$ open facility at p (cost: $2^u f + f$)
end
else
  `//` u_i `is not covered`
  With $p_{uncov} = \frac{1}{2^i}$, $\forall i \in [M]$ open facility at p (cost: $2^i f + f$)
end

6.5 Competitive Ratio

In this section, we will study the competitive ratio of the algorithm above. As someone might expect, the competitive ratio changes according to M, as we will prove in theorem 6.5.1. We will see that when $M > \frac{\log n}{\log\log n}$, it dominates in a way the competitive ratio, which then becomes $O(M)$, while when $M \leq \frac{\log n}{\log\log n}$ it is the $\frac{\log n}{\log\log n}$ that dominates. This is to be expected since when M is small, f_3 grows, so the assignment cost becomes more expensive and since OPT cannot open large clusters, the problem becomes essentially facility location in many balls of radius f. On the other hand, when M is large, it means that the assignment cost is cheaper, so we can open larger clusters, and in the extreme case where $M > \log n$, we essentially have the Sum Radii problem with the $O(\log n)$ competitive ratio.

Theorem 6.5.1. *The competitive ratio of the algorithm is:* $max\{\frac{\log n}{\log\log n}, \min\{\log n, M\}\}$

Let c_k be a simple cluster of radius 2^k. The optimal assignment cost in this cluster is Asg^*, and therefore its optimal cost is $OPT = f + 2^k f + f_3 Asg^*$. We will bound separately the expected cost of the algorithm for demands in the cases they are uncovered and covered the moment they arrive (lemmas 6.5.2 and 6.5.3).

Lemma 6.5.2. *The expected cost of uncovered demands is* $\mathbb{E}[cost_{uncov}] \leq 2^{k+1}(f(M+4)+f)+f$

Proof. We will bound the expected cost of the algorithm until they open a facility of radius 2^{k+1} which will cover all c_k. The analysis is similar to [19].

For a sequence of demands $u_1, \ldots, u_m$ we define X_i to be the random variable equal to the cost the uncovered demand i will pay. Recall from algorithm 15 that for each demand we try $i \in [M]$ times with probability $\frac{1}{2^i}$ to open facility of radius 2^i. Therefore:

$$\mathbb{E}[X_i] = \sum_{i=0}^{M} \frac{1}{2^i}(f + f2^i) = \sum_{i=0}^{M} f + \frac{f}{2^i} = f(M+2) + f(2 - 2^{-(M+1)}) \leq f(M+4)$$

We also define T as the time when a facility of radius 2^{k+1} opens. Clearly T is a stopping time since $T \in \mathbb{N}$ and depends only on the previous values of X_i. Also $T \sim \mathrm{G}_0\left(2^{-(k+1)}\right)$ thus $\mathbb{E}[T] = 2^{k+1}$

The expected cost until time T is $\mathbb{E}[cost] = \mathbb{E}[\sum_{i=0}^{T} X_i]$. We observe that X_is are iid with the same mean ($\mathbb{E}[X]$) and T has finite expectation, so we can use Wald's equation to get:

$$\mathbb{E}[cost] = \mathbb{E}[X]\mathbb{E}[T] \leq 2^{k+1} f(M+4)$$

Adding the cost for the large cluster, the total cost is:

$$cost_{uncov} \leq 2^{k+1}(f(M+4) + f) + f \tag{6.1}$$

■

This implies that uncovered demands pay something directly proportional to the inverse of the scale parameter f_3. In the second lemma, we will bound the cost paid by covered demands.

Lemma 6.5.3. *The covered demands' cost is at most* $18 + \frac{\log n}{\log \log n}$

Proof. The proof of this lemma is a somewhat more complex version of Meyerson's proof in the case of adversarial input. Since when we consider only covered demands, the restriction of radius does not exist, therefore our problem is essentially Facility Location with an extra radius cost.

Let d_u^* be the optimal assignment cost of demand u, F_u be the demand that covers u and has radius R and $\delta = d(F_u, u)$ be the distance of u from F_u.

We divide the optimal cluster into zones (as seen in figure 6.2). Each zone S_a ($a \geq 1$) contains all the demands u such that $2^{a-1} f \leq d_u^* \leq 2^a f$. Zone S_0 contains demands that $0 \leq d_u^* \leq f$. We will bound the expected cost paid by demands in each zone. Since each demand opens a facility with probability $\frac{\delta_i}{R_j/2f+f}$ the expected cost paid is:

$$\mathbb{E}[cost\ of\ u] = \frac{\delta}{R_j/2f+f} \cdot (\frac{R_j}{2} f + f) + (1 - \frac{\delta_i}{R_j/2f+f})\delta \leq 2\delta$$

Let a demand $u \in S_a$ and is covered by facility $F_u \in S_b$ with radius R. We will bound the cost of each zone, until a facility opens in this zone that covers it. After that, the

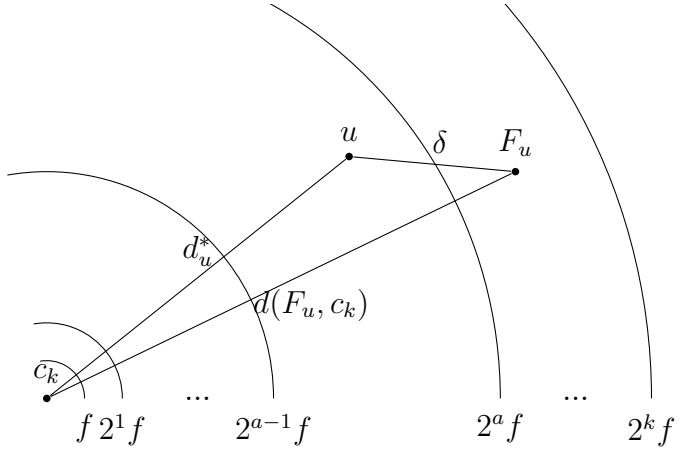

Figure 6.2: Proof for covered demands

cost will be within constant of Asg^*. We distinguish the following cases according to the facility's zone's position:

- $b \leq a + 2$ (type 1)
  In this case, the facility is either in a previous zone or at most 2 zones after the zone of the demand. If the facility's zone is one of the next 2 zones, it is not guaranteed that if a demand opens a facility, its radius will be large enough to cover zone S_a. Since $d(F_u, c_k) \leq 2^{a+2} f = 8 \cdot 2^{a-1} f = 8 d_u^*$, from the triangle inequality we get: $d(F_u, u) \leq d_u^* + d(F_u, c_k) \leq 9 d_u^*$.

  Therefore, the expected cost paid is bounded by $2\delta \leq 18 d_u^*$ and summing up for all zones, we get $\mathbb{E}[\textit{cost type } 1] \leq 18 Asg^*$

- $a + 2 < b$ (type 2)
  In this case, we get that $\delta \geq 2^{a+2} - 2^a = 3 \cdot 2^a \geq 2^{a+1}$, but since the facility covers the demand: $R \geq \delta$. Therefore if one of these demands opens a facility, the demand's zone (and all the previous zones) will be covered.

  The intuition for this part of the proof is that we cannot pay the optimal cluster $(2^k f + f)$ too many times, since from the way the algorithm works, the facilities will open every time closer to the center and with half the radius than before. This is true for all but the closest zone to c_k (where the radii do not reduce in cost) which will be considered separately.

Let F_u be the closest facility to the center. The expected cost paid by demands covered by F_u to open a facility (of radius $R/2$) is $R/2f + f$, so in total $2(R/2f + f)$ after the facility opens. This facility will open closer to the optimal center (since we care about demands between c_k and F_u). We distinguish in two cases based on whether F_u covers all previous zones or not. We will see that these two cases are not very different in the end.

- $R \geq 2^b$ (demands of zone S_a are covered all by F_u)
  In this case, since this is the closest facility to c_k, all demands of previous zones are connected to this, and will pay $\mathbb{E}[cost] = 2(R/2f + f)$ towards opening a facility of radius $R/2$, which will open in a zone closer to the center than F_u.

  Therefore, after the algorithm opens a facility in a zone that covers all the smaller ones, the new facilities will open always closer to the center and with half the radius. Summing up in this expected cost in each zone, in the worst case we will get: $\sum_{i=0}^{k} 2(2^i f + f) \leq 2\sum_{i=0}^{k} 2^{i+1} f = 8 \cdot 2^k f$

- $R < 2^b$ (the demands of the zone S_a are covered by facilities other than F_u)
  This case is in fact similar to the previous one since only the radius of the facility covering the demands matters for the cost, and not the position of the facility.

  More specifically, since all facilities are more than 2 zones away, every new facility will cover previous zones, so we will pay at most the expected cost for the largest one that covers the demands, to open a new one. After this opens, we are within optimal as shown in case $a + 2 < b$

So $\mathbb{E}[cost\ type\ 2] \leq 8 \cdot 2^k f$

The above analysis does not hold for the S_0 where the algorithm reduces to Meyerson's online Facility Location ([4]) with facility cost $2f$. This algorithm was shown in [5] to be asymptotically optimal, therefore the competitive ratio is $\frac{\log n}{\log \log n}$.

We proved that totally, the cost of the covered demands is within $18 + \frac{\log n}{\log \log n}$ of the optimal cost.

■

Now we will prove the main theorem.

Proof of theorem 6.5.1. Since we allow the opening of clusters inside larger clusters, it is possible that the optimal cluster will have other smaller clusters inside it. We say that a cluster is simple, if it has no other clusters inside it.

Using Lemmas 6.5.2 and 6.5.3 we bound the optimal cost for any simple cluster by $\max\{M, \frac{\log n}{\log\log n}\}$, since the M factor comes from uncovered demands, and the $\frac{\log n}{\log\log n}$ factor from the covered. Therefore, in a larger cluster we bound the cost considering only the demands assigned to the large. The cost of the demands in the smaller ones, will have been charged to the optimal cost for the smaller ones.

We should be careful though, since the competitive ratio cannot be more than $\log n$. This is easily seen from lemma 6.5.2 ; if $M > \log n$, the assignment cost count so little, that it can always be charged to the facility opening cost, so our problem degenerated to the Sum Radii problem with essentially the same algorithm as [19]. ■

Observe that in the case we are closer to Facility Location, the algorithm is tight: we achieve the $\frac{\log n}{\log\log n}$ lower bound. On the other side however, when the problem is closer to the Sum Radii one, we do not know whether it gives the best possible competitive ratio.

Chapter 7

Open Problems, Future Work

Continuing the work on this variant, it would be interesting to study the problem in the case where each potential facility has a different opening cost f_i, namely the non uniform facility costs variant. In this case, the scale factor f_3 would not be necessary, since the assignment, radius and opening cost would not be of the same order, which was causing the limitations in the optimal in our case with $f_3 = 1$.

Additionally, it is worth mentioning that another, slightly better motivated, variant is when we allow clients to connect to facilities outside their radius, and pay the connection cost, while the clients inside the facilities would pay nothing. We can think of the demands as clients who want internet access, but have no restriction as to what type of access they will be provided with (cable or wireless). In this case however, we run into the same problem as before: since the opening, assignment and radius costs will be of the same order, we will need a scale factor for the assignment cost, in order for the problem to be non trivial. It seems that generally in the variants where we impose both a connection and a radius cost, we will always run into the same problem.

Another interesting direction would be to resolve the open question of [19], namely close the gap of the Sum Radii algorithm ($\log n$) and the lower bound of $\log \log n$ for randomized algorithms, which will also prove whether our algorithm is optimal or not, in the case we are closer to the Sum Radii problem.

An interesting direction, though quite different from the *online* setting we discussed before, would be to assume a distribution on the input data instead of trying to find the worst case one. In some problems, the input indeed is drawn from a distribution, so this is sometimes a setting closer to the real world than the worst case one.

Finally, since it is clear that Facility Location and its variants have numerous applications, an interesting direction would be to consider some of the other offline Facility Location variants, like capacitated or fault tolerant Facility Location in the online or incremental settings.

Bibliography

[1] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, pages 77–88, 2011.

[2] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999.

[3] Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

[4] Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 426–431, 2001.

[5] Dimitris Fotakis. Online and incremental algorithms for facility location. *SIGACT News*, 42(1):97–131, 2011.

[6] Dimitris Fotakis. A primal-dual algorithm for online non-uniform facility location. *J. Discrete Algorithms*, 5(1):141–148, 2007.

[7] Aris Anagnostopoulos, Russell Bent, Eli Upfal, and Pascal Van Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194(2):175–202, 2004.

[8] Jaroslaw Byrka. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop, APPROX 2007, and 11th International Workshop, RANDOM 2007, Princeton, NJ, USA, August 20-22, 2007, Proceedings*, pages 29–43, 2007.

[9] David B. Shmoys, Éva Tardos and Karen Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the Twenty-Ninth*

Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997, pages 265–274, 1997.

[10] Fabián A. Chudak. Improved approximation algorithms for uncapacitated facility location (extended abstract). pages 180–194. Springer, 1998.

[11] Maxim Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In William J. Cook and Andreas S. Schulz, editors, *Integer Programming and Combinatorial Optimization*, pages 240–257. Springer Berlin Heidelberg, 2002.

[12] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.

[13] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 731–740, 2002.

[14] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *J. ACM*, 50(6):795–824, 2003.

[15] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation algorithms for metric facility location problems. *SIAM J. Comput.*, 36(2):411–432, 2006.

[16] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 378–388, 1999.

[17] Srinivas Doddi , Madhav Marathe , S. Ravi , David Scot Taylor , Peter Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. In *Algorithm Theory - SWAT 2000*, pages 237–250, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[18] Moses Charikar, Rina Panigrahy. Clustering to minimize the sum of cluster diameters. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 1–10, 2001.

[19] Dimitris Fotakis and Paraschos Koutris. Online sum-radii clustering. *Theor. Comput. Sci.*, 540:27–39, 2014.

[20] Dimitris Fotakis. Incremental algorithms for facility location and k-median. *Theor. Comput. Sci.*, 361(2-3):275–313, 2006.

[21] Jyh-Han Lin and Jeffrey Scott Vitter. Approximation algorithms for geometric median problems. *Inf. Process. Lett.*, 44(5):245–249, 1992.

[22] Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.

[23] Ramgopal R. Mettu and C. Greg Plaxton. The online median problem. *SIAM J. Comput.*, 32(3):816–832, 2003.

[24] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[25] Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 250–257, 2002.

[26] Moses Charikar, Chandra Chekuri, Tomás Feder, Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

[27] Chandan K. Reddy and Bhanukiran Vinzamuri. A survey of partitional and hierarchical clustering algorithms. In *Data Clustering: Algorithms and Applications*, pages 87–110. 2013.

[28] L.G.Khachiyan. Polynomial algorithms in linear programming. *USSR Comput. Maths Math. Phys*, 20(1):53–72, 1980.

[29] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

[30] Howard Karlof. *Linear Programming*. Birkhauser, 1991.

[31] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.

[32] Carsten Lund , Mihalis Yannakakis. On the hardness of approximating minimization problems. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 286–293, 1993.

[33] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[34] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[35] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[36] Susanne Albers. Online algorithms: a survey. *Math. Program.*, 97(1-2):3–26, 2003.

[37] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

[38] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[39] Tarjan Robert. Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6(2):306–318, 1985.

[40] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in online algorithms (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 379–386, 1990.

[41] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227, 1977.

[42] Boris Teia. A lower bound for randomized list update algorithms. *Inf. Process. Lett.*, 47(1):5–9, 1993.

[43] Susanne Albers, Bernhard von Stengel, and Ralph Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Inf. Process. Lett.*, 56(3):135–139, 1995.

[44] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 322–333, 1988.

[45] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.

[46] Marek Chrobak and Lawrence L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, February 11-13, 1991*, pages 11–64, 1991.

[47] Howard J. Karloff, Yuval Rabani, and Yiftach Ravid. Lower bounds for randomized k-server and motion planning algorithms. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, pages 278–288, 1991.

[48] James R. Lee. Fusible HSTs and the randomized k-server conjecture. *CoRR*, abs/1711.01789, 2017.

[49] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal online algorithm for metrical task systems. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 373–382, 1987.