National Technical University of Athens
School of Electrical and Computer Engineering

Online Algorithms for Dynamic Aggregation Problems

PhD Thesis

Loukas Kavouras

Athens, Greece
May 2021

This research is co-financed by Greece and the European Union (European Social Fund- ESF) through the Operational Programme «Human Resources Development, Education and Lifelong Learning» in the context of the project "Scholarships programme for post-graduate studies - 2nd Study Cycle" (MIS-5003404), implemented by the State Scholarships Foundation (IKY).

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Άμεσοι Αλγόριθμοι για Προβλήματα Δυναμικής Συνάθροισης

Διδακτορική Διατριβή

Λουκάς Κάβουρας

Αθήνα,
Μάιος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Άμεσοι Αλγόριθμοι για Προβλήματα Δυναμικής Συνάθροισης

Λουκάς Κάβουρας

Τριμελής Συμβουλευτική Επιτροπή:
Δημήτρης Φωτάκης *(επιβλέπων)*
Ευστάθιος Ζάχος
Αριστείδης Παγουρτζής

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την **27/05/2021**:

Δημήτριος Φωτάκης	**Αριστείδης Παγουρτζής**	**Ιωάννης Εμίρης**
Αν.Καθηγητής, ΕΜΠ	Καθηγητής, ΕΜΠ	Καθηγητής, ΕΚΠΑ
Αντώνιος Συμβώνης	**Σπυρίδων Κοντογιάννης**	
Καθηγητής, ΕΜΠ	Αν.Καθηγητής, Παν. Ιωαννίνων	
Βασίλης Ζησιμόπουλος	**Γιώργος Στάμου**	
Καθηγητής, ΕΚΠΑ	Αν. Καθηγητής, ΕΜΠ	

Αθήνα, Μάιος 2021

. . .

Λουκάς Κάβουρας
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Σε αυτή την διδακτορική διατριβή, μελετάμε άμεσους αλγόριθμους για προβλήματα Δυναμικής Συνάθροισης. Στην περίπτωση των άμεσων αλγόριθμων θεωρούμε ότι η είσοδος των προβλημάτων δεν είναι γνωστή εκ των προτέρων, αλλά αποκαλύπτεται κομμάτι-κομμάτι στον αλγόριθμο. Τα αποτελέσματα μας αφορούν άμεσες εκδοχές των προβλημάτων Κάλυψης Συνόλου Ελάχιστου Αθροίσματος , Ανακατανομής Κ-Υπηρεσιών και Δυναμικής Χωροθέτησης Υπηρεσιών. Για όλα αυτά τα προβλήματα σχεδιάζουμε άμεσους αλγόριθμους και αποδεικνύουμε την αποτελεσματικότητά τους. Συγκεκριμένα, αποδεικνύουμε άνω φράγματα στον λόγο ανταγωνιστικότητας των άμεσων αλγορίθμων μας. Ο λόγος ανταγωνιστικότητας ορίζεται ως ο χειρότερος δυνατό λόγος ανάμεσα στο κόστος της λύσης του άμεσου αλγορίθμου και στο κόστος της λύσης ενός βέλτιστου αλγορίθμου, ο οποίος επιπλέον γνωρίζει όλη την είσοδο εκ των προτέρων. Επιπλέον, σχεδιάζουμε δύσκολα στιγμιότυπα για όλα τα προβλήματα που μελετάμε, τα οποία μας δίνουν την δυνατότητα να αποδείξουμε κάτω φράγματα στον καλύτερο δυνατό λόγο ανταγωνιστικότητας που μπορεί να επιτύχει οποιοσδήποτε άμεσος αλγόριθμος. Αυτό σημαίνει ότι κανένας αλγόριθμος δεν μπορεί να πετύχει καλύτερο λόγο ανταγωνιστικότητας από το κάτω φράγμα που ισχύει για τα δύσκολα στιγμιότυπα. Στις περισσότερες περιπτώσεις, τα κάτω φράγματα είναι πολύ κοντά στα άνω φράγματα (σε κάποιες περιπτώσεις ταυτίζονται) και το γεγονός αυτό αποδεικνύει ότι οι αλγόριθμοι που έχουμε διατυπώσει και αναλύσει είναι βέλτιστοι ή σχεδόν βέλτιστοι για το αντίστοιχο πρόβλημα.

Abstract

In this Ph.D thesis, we study online variants of Dynamic Aggregation problems that are generalizations of prominent and well studied online problems. In the online setting, we additionally assume that the input arrives piece-by-piece and that the online algorithm has to provide a solution for the input piece of the current stage before it sees the upcoming input pieces of future stages. The decision quality of the online algorithm is evaluated against an optimal offline algorithm, which is given the whole problem data from the beginning. The performance of the online algorithm is measured by the competitive ratio which is the worst-case ratio between the online cost and the optimal offline cost. We consider the online variants of the Min-Sum Set Cover problem, the K-Facility Reallocation problem and the Dynamic Facility Location problem. For all the aforementioned problems, we design online algorithms and we prove upper bounds on their competitive ratio. Moreover, we construct difficult instances for these problems and we prove lower bounds on the competitive ratio of online algorithms on these instances. The majority the upper bounds are close (or the same) with the lower bounds that we prove and this ensures that our online algorithms are optimal or near optimal.

Ευχαριστίες

Η παρούσα διδακτορική διατριβή δεν θα μπορούσε να είχε πραγματοποιηθεί χωρίς την συμβολή του επιβλέποντα μου κ. Δημήτρη Φωτάκη. Τον ευχαριστώ από καρδιάς για τη βοήθεια που μου προσέφερε τόσο σε επιστημονικό όσο και σε προσωπικό επίπεδο, αλλά και για όλο το ενδιαφέρον, την υπομονή και την καθοδήγηση του. Σημαντικές ευχαριστίες οφείλονται και στα άλλα δύο μέλη της τριμελούς συμβουλευτικής επιτροπής μου, δηλαδή τους κ. Στάθη Ζάχο και Άρη Παγουρτζή για την συμπαράσταση και την βοήθεια που μου έχουν προσφέρει αλλά και για το άριστο περιβάλλον που έχουν δημιουργήσει μαζί με τον κ. Δημήτρη Φωτάκη στο εργαστήριο Corelab. Ιδιαίτερες ευχαριστίες απευθύνονται στον κ. Ιωάννη Εμίρη, με τον οποίο πραγματοποίησα την πρώτη δημοσίευσή μου στα πλαίσια του μεταπτυχιακού ΜΠΛΑ. Με τιμά ιδιαίτερα επίσης η συμμετοχή των κ.κ. Συμβώνη, Κοντογιάννη, Ζησιμόπουλου και Στάμου στην επταμελή επιτροπή εξέτασης της διατριβής μου, τους οποίους και ευχαριστώ. Πολλά ευχαριστώ σε όλους τους συναδέλφους εντός και εκτός Corelab για την στήριξη, τις συνεργασίες και για όλες τις ομόρφες και δύσκολες στιγμές που περάσαμε. Ιδιαίτερα θα ήθελα να ευχαριστήσω τον Στρατή Σκουλάκη και τον Γρηγόρη Κουμούτσο. Σημαντικές ευχαριστίες οφείλω και στο Ίδρυμα Κρατικών Υποτροφιών (ΙΚΥ) για την χρηματοδότηση της διδακτορικής μου διατριβής. Τέλος, θέλω να ευχαριστήσω την οικογένεια μου για την στήριξη και την κατανόηση σε όλη αυτή την ωραία αλλά και δύσκολη διαδρομή.

Εκτεταμένη Περίληψη

Σε αυτήν τη διδακτορική διατριβή, μελετάμε παραλλαγές *άμεσων* προβλημάτων Δυναμικής Συνάθροισης που είναι γενικεύσεις διακεκριμένων και καλά μελετημένων άμεσων προβλημάτων. Η αντικειμενική συνάρτηση των προβλημάτων Δυναμικής Συνάθροισης περιλαμβάνει μια συνάρτηση κόστους Συνάθροισης με τυπικά παραδείγματα τις συναρτήσεις Ελαχίστου, Μεγίστου, Μέσου Όρου και Αθροίσματος. Κάθε πρόβλημα Δυναμικής Συνάθροισης θεωρούμε ότι έχει μια δυναμική εξέλιξη στον χρόνο, η οποία πραγματοποιείται σε Στάδια. Αυτή η δυναμική εξέλιξη αποτυπώνεται και στην αντικειμενική συνάρτηση ενός προβλήματος Δυναμικής Συνάθροισης, η οποία περιέχει επιπλέον ένα κόστος Εναλλαγής (ή Μετακίνησης) για μια λύση που διαφέρει από τη λύση του προηγούμενου σταδίου. Κατά συνέπεια, οι αλγόριθμοι για προβλήματα Δυναμικής Συνάθροισης στοχεύουν στην ανακάλυψη της χρονικής εξέλιξης στοιχείων, τα οποία δεν είναι πολύ ευαίσθητα σε παροδικές αλλαγές.

Αν θεωρήσουμε επιπλέον ότι η είσοδος του προβλήματος Δυναμικής Συνάθροισης δεν είναι γνωστή εκ των προτέρων, αλλά αποκαλύπτεται κομμάτι-κομμάτι σε κάθε στάδιο, τότε μελετάμε την *άμεση* εκδοχή ενός προβλήματος Δυναμικής Συνάθροισης. Οι άμεσες εκδοχές προβλημάτων Δυναμικής Συνάθροισης προκύπτουν σε πολλές πρακτικές εφαρμογές προερχόμενες από διαφορετικούς κλάδους όπως η επιδημιολογία, ο σχεδιασμός εμβολιασμού, ο πολεοδομικός σχεδιασμός, τα κοινωνικά δίκτυα και οι μηχανές αναζήτησης. Αυτές οι εφαρμογές περιλαμβάνουν έναν ταχέως αυξανόμενο όγκο διαθέσιμων δεδομένων που υπόκεινται σε γρήγορες και μη προβλέψιμες αλλαγές. Σε αντίθεση με την φύση αυτών των πρακτικών εφαρμογών, η οποία είναι εγγενώς άμεση, η επικρατούσα προσέγγιση για προβλήματα Δυναμικής Συνάθροισης στην επιστημονική βιβλιογραφία προϋποθέτει την πλήρη γνώση των εισόδων που θα αποκαλυφθούν σε κάθε στάδιο στον αλγόριθμο. Το βασικό κίνητρο και ο σκοπός της παρούσας εργασίας είναι να εξερευνήσει προβλήματα Δυναμικής Συνάθροισης σε πιο ρεαλιστικές περιπτώσεις, κατά τις οποίες δεν είναι δυνατό να προβλεφθούν οι μελλοντικές είσοδοι και αναζητούνται αποδοτικές λύσεις δεδομένης της ύπαρξης αυτής της αβεβαιότητας.

Ο τομέας που ασχολείται με την ανάπτυξη αλγορίθμων, οι οποίοι χειρίζονται καταστάσεις αβεβαιότητας, ονομάζεται Άμεση Βελτιστοποίηση και οι αλγόριθμοι που έχουν αυτά τα χαρακτηριστικά ονομάζονται άμεσοι αλγόριθμοι. Ένας άμεσος αλγόριθμος έχει τη δυνατότητα να επεξεργάζεται την είσοδο κομμάτι-κομμάτι με τη σειρά που αποκαλύπτεται και υποχρεούται να εξυπηρετήσει το κομμάτι της εισόδου που του έχει αποκαλυφθεί μια δεδομένη χρονική στιγμή πριν του αποκαλυφθεί το επόμενο κομμάτι εισόδου. Επιπλέον, η απόφαση για το πως θα εξυπηρετήσει ένα κομμάτι της εισόδου είναι *αμετάκλητη*, δηλαδή δεν μπορεί να αλλάξει στο μέλλον και όταν θα έχει δει τα επόμενα μέρη της εισόδου. Αυτός ο περιορισμός οδηγεί τους άμεσους αλγορίθμους σε μη *βέλτιστες* αποφάσεις επηρεάζοντας την *αποδοτικότητά* τους. Για να μετρήσουμε την αποδοτικότητα ενός άμεσου αλγορίθμου,

χρησιμοποιούμε μεθόδους από την Ανταγωνιστική Ανάλυση και συγκεκριμένα τον *λόγο ανταγωνιστικότητας*, ο οποίος είναι ο χειρότερος δυνατός λόγος μεταξύ του κόστους ενός άμεσου αλγορίθμου και του κόστους ενός βέλτιστου αλγορίθμου, ο οποίος γνωρίζει όλη την είσοδο εκ των προτέρων. Η βασική επιδίωξη στο πεδίο της Άμεσης Βελτιστοποίησης είναι η ανάπτυξη αλγορίθμων με βέλτιστο λόγο ανταγωνιστικότητας και η κατασκευή κάτω φραγμάτων στους λόγους ανταγωνιστικότητας των άμεσων αλγορίθμων, τα οποία αποτελούν την καλύτερη δυνατή επίδοση που μπορεί να επιτύχει οποιοσδήποτε άμεσος αλγόριθμος (άρα και ο πιο αποδοτικός από τους άμεσους αλγορίθμους) για ένα συγκεκριμένο πρόβλημα.

Η κύρια συνεισφορά της παρούσας επιστημονικής διατριβής είναι η κατασκευή άμεσων αλγορίθμων για τα προβλήματα Κάλυψης Συνόλου Ελάχιστου-Αθροίσματος, Ανακατανομής K-Υπηρεσιών και Δυναμικής Χωροθέτησης Υπηρεσιών. Όλοι οι άμεσοι αλγόριθμοι που παρουσιάζουμε πετυχαίνουν βέλτιστους ή σχεδόν βέλτιστους λόγους ανταγωνιστικότητας για τα προβλήματα που εξετάζουμε, καθώς για όλα τα προβλήματα αποδεικνύουμε επιπλέον κάτω φράγματα στον λόγο ανταγωνιστικότητας, τα οποία δεν απέχουν πολύ (ή είναι ίδια) από τα άνω φράγματα των άμεσων αλγορίθμων μας. Επιπλέον, εξερευνούμε τη σύνδεση των παραπάνω προβλημάτων με γνωστά και καλά μελετημένα άμεσα προβλήματα, από τα οποία αντλούμε ιδέες για την κατασκευή των άμεσων αλγορίθμων μας αλλά και για την απόδειξη *εγγυήσεων* στην αποδοτικότητά τους όσο αφορά τον λόγο ανταγωνιστικότητας. Τέλος, η σύνδεση αυτή μας επιτρέπει να κατασκευάσουμε κάτω φράγματα στον λόγο ανταγωνιστικότητας για κάποια προβλήματα μέσω *αναγωγής* σε κάποιο πρόβλημα για το οποίο υπάρχει γνωστό κάτω φράγμα.

Στο πρώτο εισαγωγικό κεφάλαιο της παρούσας εργασίας (1), δίνουμε τον ορισμό των προβλημάτων Δυναμικής Συνάθροισης και παρουσιάζουμε παραδείγματα πολλών εφαρμογών από διαφορετικούς επιστημονικούς τομείς. Στη συνέχεια, εξερευνούμε την πλούσια επιστημονική βιβλιογραφία που αφορά δημοφιλή άμεσα προβλήματα Δυναμικής Συνάθροισης όπως το πρόβλημα των K-Εξυπηρετητών [54],[55],[60],[41] και το πρόβλημα Προσπέλασης Λίστας [3], [66], [1],[2] και σχολιάζουμε την σχέση τους με τα προβλήματα Κάλυψης Συνόλου Ελάχιστου-Αθροίσματος, Ανακατανομής K-Υπηρεσιών και Δυναμικής Χωροθέτησης Υπηρεσιών. Ακολουθεί η παρουσίαση των προβλημάτων Κάλυψης Συνόλου Ελάχιστου-Αθροίσματος, Ανακατανομής K-Υπηρεσιών και Δυναμικής Χωροθέτησης Υπηρεσιών και η παράθεση της σχετικής βιβλιογραφίας καθώς των αποτελεσμάτων μας για αυτά τα προβλήματα.

Η τελευταία ενότητα του εισαγωγικού κεφαλαίου περιέχει το απαραίτητο τεχνικό υπόβαθρο για τις έννοιες της Άμεσης Βελτιστοποίησης [23] που χρησιμοποιούμε στην υπόλοιπη έκταση του κειμένου. Αρχικά, παρουσιάζουμε την έννοια της Ανταγωνιστικής Ανάλυσης [23], η οποία χρησιμοποιείται στην βιβλιογραφία για την ανάλυση της αποδοτικότητας των άμεσων αλγορίθμων. Στην Ανταγωνιστική Ανάλυση, υποθέτουμε την ύπαρξη ενός

κακού *αντιπάλου*, ο οποίος δημιουργεί εισόδους του προβλήματος και τις αποκαλύπτει μία μία στον άμεσο αλγόριθμο. Όπως προαναφέραμε, υποχρέωση του άμεσου αλγορίθμου είναι να εξυπηρετήσει το κομμάτι της εισόδου την ίδια στιγμή και δεν μπορεί να αλλάξει αυτή την απόφαση στο μέλλον. Στη συνέχεια ο αντίπαλος θα αποκαλύψει το επόμενο κομμάτι της εισόδου. Όσο αφορά τους ντετερμινιστικούς αλγορίθμους, θεωρούμε ότι ο αντίπαλος γνωρίζει τον κώδικα τους και τους δίνει τις χειρότερες δυνατές εισόδους με σκοπό να αυξήσει το συνολικό κόστος εξυπηρέτησης των εισόδων. Όσο αφορά τους πιθανοτικούς αλγορίθμους, έχουμε τρία διαφορετικά μοντέλα αντιπάλων που αναλύονται διεξοδικά στην ενότητα 1.3.1.

Στο δεύτερο κεφάλαιο (2), συνοψίζουμε αλγοριθμικές μεθόδους που έχουμε χρησιμοποιήσει για να σχεδιάσουμε και αναλύσουμε άμεσους αλγορίθμους, καθώς και μεθόδους που μας βοήθησαν στην κατασκευή κάτω φραγμάτων στους λόγους ανταγωνιστικότητας. Ξεκινάμε με την μέθοδο Μέθοδο Ενημέρωσης Πολλαπλασιαστικών Βαρών [58],[9], την οποία χρησιμοποιούμε για να σχεδιάσουμε αλγόριθμο με βέλτιστο λόγο ανταγωνιστικότητας για το πρόβλημα Κάλυψης Συνόλου Ελάχιστου-Αθροίσματος. Στη συνέχεια, παρουσιάζουμε την Μέθοδο Δυνητικής Συνάρτησης [23],[66], την οποία εκμεταλλευόμαστε για να αποδείξουμε άνω φράγματα σε λόγους ανταγωνιστικότητας όσο αφορά το πρόβλημα Κάλυψης Συνόλου Ελάχιστου-Αθροίσματος και το πρόβλημα Ανακατανομής K-Υπηρεσιών. Η Μέθοδος Δυνητικής Συνάρτησης χρησιμοποιείται ευρέως στην ανάλυση τους κόστους των άμεσων αλγορίθμων, όμως η επιλογή της κατάλληλης Δυνητικής Συνάρτησης απαιτεί βαθιά γνώση των ιδιοτήτων του εκάστοτε προβλήματος αλλά και διαίσθηση. Ακολουθεί η μέθοδος της Κανονικοποίησης [12], η οποία χρησιμοποείται ευρέως στα πεδία της Κυρτής Βελτιστοποίησης και της Μάθησης. Εμείς την χρησιμοποιούμε για την κατασκευή ενός πιθανοτικού άμεσου αλγορίθμου για το πρόβλημα Δυναμικής Χωροθέτης Υπηρεσιών. Τέλος, εκθέτουμε μεθόδους για την κατασκεύη κάτω φραγμάτων στον λόγο ανταγωνιστικότητας άμεσων αλγορίθμων όπως είναι η Αρχή του Yao [23] και ο Υπολογισμός Μέσου Όρου Αντιπάλων [55].

Στα επόμενα τρία κεφάλαια της διδακτορικής διατριβής παρουσιάζουμε και αναλύουμε τα αποτελέσματά μας για τις άμεσες εκδοχές των προβλημάτων Κάλυψης Συνόλου Ελάχιστου Αθροίσματος, Ανακατανομής K-Υπηρεσιών και Δυναμικής Χωροθέτησης Υπηρεσιών.

Κάλυψη Συνόλου Ελάχιστου Αθροίσματος

Ξεκινάμε με το πρόβλημα Κάλυψης Συνόλου Ελάχιστου-Αθροίσματος, το οποίο αποτελεί μια φυσιολογική και ενδιαφέρουσα γενίκευση του προβλήματος Προσπέλασης Λίστας. Στην άμεση εκδοχή του προβλήματος Κάλυψης Συνόλου Ελάχιστου-Αθροίσματος, ο αλγόριθμος διατηρεί μια μετάθεση σε n στοιχεία με βάση τα υποσύνολα $S_1, S_2, \ldots$, τα οποία αποκαλύπτονται ένα ένα σε κάθε στάδιο. Ο αλγόριθμος εξυπηρετεί κάθε σύνολο S_t κατά

την άφιξη του, χρησιμοποιώντας την τρέχουσα μετάθεση του p_t, και πληρώνει ένα κόστος *Πρόσβασης* ίσο με τη θέση του πρώτου στοιχείου S_t στη μετάθεση p_t. Στη συνέχεια, ο αλγόριθμος μπορεί υπολογίσει μια καινούρια μετάθεση p_{t+1}, με κόστος *Μετακίνησης* ίσο με την απόσταση Kendall tau μεταξύ της p_t και της p_{t+1}, δηλαδή των αριθμό των ελάχιστων αναστροφών στοιχείων που πρέπει να πραγματοποιηθούν σε μία από τις δύο έτσι ώστε να γίνει ίδια με την άλλη. Ο στόχος είναι να ελαχιστοποιηθεί το συνολικό κόστος *Πρόσβασης* και το συνολικό κόστος *Μετακίνησης* για την εξυπηρέτηση ολόκληρης της ακολουθίας των υποσυνόλων $S_1, S_2, \ldots$. Θεωρούμε, χωρίς βλάβη της γενικότητας, ότι κάθε S_t έχει ακριβώς r στοιχεία και μελετάμε αυτή την περίπτωση. Το πρόβλημα Προσπέλασης Λίστας είναι η ειδική περίπτωση όπου $r = 1$.

Συγκρίνουμε τον λόγο ανταγωνιστικότητας των άμεσων αλγορίθμων μας για το προβλήμα της Κάλυψης Συνόλου Ελάχιστου Αθροίσματος με δύο διαφορετικούς βέλτιστες λύσεις, οι οποίες δίνονται από αλγορίθμους που γνωρίζουν εκ των προτέρων την ακολουθία $S_1, S_2, \ldots$ και την εξυπηρετούν βέλτιστα. Ο πρώτος αλγόριθμος, τον οποίο ονομάζουμε *στατικό* βέλτιστο αλγόριθμο, υπολογίζει την βέλτιστη λύση δεδομένου ότι διατηρεί μία μόνο μετάθεση p . Ο δεύτερος αλγόριθμος, τον οποίο ονομάζουμε *δυναμικό* βέλτιστο αλγόριθμο, υπολογίζει την βέλτιστη λύση έχοντας τη δυνατότητα να αλλάζει την μετάθεση του μεταξύ των σταδίων. Ουσιαστικά, ο δυναμικός βέλτιστος αλγόριθμος υπολογίζει την βέλτιστη ακουλουθία μεταθέσεων $p_1, p_2, \ldots$ για την εξυπηρέτηση των υποσυνόλων $S_1, S_2, \ldots$. Είναι εύκολο να αποδείξει κανείς ότι υπάρχουν στιγμιότυπα του προβλήματος, για τα οποία ο βέλτιστος δυναμικός αλγόριθμος μπορεί να εξυπηρετήσει την ακολουθία των υποσυνόλων $S_1, S_2, \ldots$ με κόστος n φορές μικρότερο από τον στατικό βέλτιστο αλγόριθμο.

Όσο αφορά την περίπτωση του βέλτιστου στατικού αλγορίθμου, αρχικά παρουσιάζουμε ένα κάτω φράγμα ίσο με $(r + 1)(1 \quad \frac{r}{n+1})$ για τον λόγο ανταγωνιστικότητας οποιουδήποτε ντετερμινιστικού άμεσου αλγορίθμου, χρησιμοποιώντας την τεχνική του Υπολογισμού Μέσου Όρου Αντιπάλων. Προφανώς αυτό το κατω φράγμα ισχύει και για την περίπτωση του δυναμικού βέλτιστου αλγορίθμου. Στη συνέχεια, εξετάζουμε αρκετές φυσιολογικές γενικεύσεις αποδοτικών άμεσων αλγορίθμων του προβλήματος Προσπέλασης Λίστας και αποδεικνύουμε ότι δεν καταφέρνουν να πετύχουν καλό λόγο ανταγωνιστικότητας ακόμα και σε πολύ απλά στιγμιότυπα, όπου τα υποσύνολα περιλαμβάνουν μόνο δύο στοιχεία ($r = 2$). Το γεγονός αυτό μας οδηγεί να στραφούμε σε μια εντελώς διαφορετική προσέγγιση για τον σχεδιασμό ενός αποδοτικού άμεσου αλγορίθμου και να εκμεταλλευτούμε τεχνικές που χρησιμοποιούνται ευρύτατα στη Άμεση Μάθηση. Το αποτέλεσμα αυτής της προσπάθειας είναι η κατασκευή του άμεσου ντετερμινιστικού αλγόριθμου Lazy Rounding με λόγο ανταγωνιστικότητας $5r + 2$, ο οποίος βασίζεται στη Μέθοδο Ενημέρωσης Πολλαπλασιαστικών Βαρών. Η συνεισφορά μας βασίζεται στο ότι καταφέρνουμε να μετατρέψουμε την πιθανοτική λύση της Μεθόδου Ενημέρωσης Πολλαπλασιαστικών Βαρών σε μια

ντετερμινιστική λύση χάνοντας μόνο έναν παράγοντα r στο κόστος Πρόσβασης σε σχέση με την πιθανοτική λύση. Επιπλέον, αποδεικνύοντας ιδιότητες που συνδέουν τις κατανομές πιθανότητας της Μεθόδου Ενημέρωσης Πολλαπλασιαστικών Βαρών με το κόστος Πρόσβασης που πληρώνει η μέθοδος, καταφέρνουμε να φράξουμε και το κόστος Μετακίνησης του Lazy Rounding και να επιτύχουμε τον επιθυμητό λόγο ανταγωνιστικότητας.

Ένα μειονέκτημα του Lazy Rounding είναι ότι έχει χρόνο εκτέλεσης $(n!)$, ο οποίος είναι εκθετικός ως προς την είσοδο του προβλήματος Κάλυψης Συνόλου Ελάχιστου Αθροίσματος. Το γεγονός αυτό μας οδήγησε στο να μελετήσουμε πιο αποδοτικούς σε χρόνο εκτέλεσης αλγορίθμους και να εξετάσουμε τον λόγο ανταγωνιστικότητας τους. Σε αυτή την κατεύθυνση, σχεδιάσαμε τον αλγόριθμο Move-All-Equally , ο οποίος σε κάθε στάδιο t μετακινεί όλα τα στοιχεία του υποσυνόλου S_t με την ίδια ταχύτητα προς την αρχή της μετάθεσης μέχρι το πρώτο από αυτά να βρεθεί στην πρώτη θέση της μετάθεσης. Μια ενδιαφέρουσα ιδιότητα του Move-All-Equally είναι ότι δεν έχει μνήμη, δηλαδή ο τρόπος που εξυπηρετεί το κάθε υποσύνολο δεν εξαρτάται από τα πως εξυπηρέτησε τα προηγούμενα υποσύνολα. Δείχνουμε ότι ο Move-All-Equally έχει κάτω φράγμα στον λόγο ανταγωνιστικότητας ίσο με $\mathsf{W}(r^2)$ και άνω φράγμα στον λόγο ανταγωνιστικότητας ίσο με $2^{O(\sqrt{\log n \log r})}$ σε σχέση με τον στατικό βέλτιστο αλγόριθμο. Η εικασία μας είναι ότι η ανάλυση για το άνω φράγμα μπορεί να βελτιωθεί έτσι ώστε να αποφευχθεί η εξάρτηση από το n.

Όσο αφορά τον δυναμικό βέλτιστο αλγόριθμο, εξετάζουμε την αποδοτικότητα του Move-All-Equally. Όπως είναι αναμενόμενο, σε αυτή την περίπτωση είναι πολύ πιο δύσκολο να πετύχουμε αποδοτικές λύσεις και αυτό αποτυπώνεται στο κάτω φράγμα για τον λόγο ανταγωνιστικότητας του Move-All-Equally, το οποίο είναι ίσο με $\mathsf{W}(r^{\overline{p}\,\overline{n}})$. Αυτό το κάτω φράγμα απαιτεί την κατασκευή ενός περίπλοκου στιγμιοτύπου, κατά το οποίο εξασφαλίζεται ότι ο Move-All-Equally πληρώνει για κάθε υποσύνολο $r^{\overline{p}\,\overline{n}}$ φορές παραπάνω από τον δυναμικό βέλτιστο αλγόριθμο. Όσο αφορά το άνω φράγμα στον λόγο ανταγωνιστικότητας του Move-All-Equally, αποδεικνύουμε χρησιμοποιώντας την Μέθοδο της Δυνητικής Συνάρτησης ότι είναι ίσο με $O(r^{3/2^{\overline{p}\,\overline{n}}})$, δηλαδή απέχει μόνο κατά ένα παράγοντα $r^{\overline{p}\,\overline{r}}$ από το κάτω φράγμα.

Ανακατανομή K-Υπηρεσιών

Στο τέταρτο κεφάλαιο μελετάμε το πρόβλημα Ανακατανομής K-Υπηρεσιών όταν ο μετρικός χώρος είναι η ευθεία των πραγματικών αριθμών. Σε αυτό το πρόβλημα, ο άμεσος αλγόριθμος καθορίζει τις θέσεις των K υπηρεσιών πάνω στην ευθεία γραμμή για ένα σύνολο T σταδίων. Η επιλογή των θέσεων γίνεται με βάση τις εξαρτώμενες από το κάθε στάδιο θέσεις ενός συνόλου n πελατών. Θεωρούμε ότι κάθε πελάτης είναι συνδεδεμένος στην πλησιέστερη σε αυτόν υπηρεσία σε κάθε στάδιο και οι υπηρεσίες μπορούν να μετακινηθούν από το ένα στάδιο στο άλλο, για να εξυπηρετήσουν καλύτερα τις διαφορετικές τοποθεσίες των πελατών. Ο στόχος είναι να ελαχιστοποιηθεί η απόσταση των πελατών

από την πλησιέστερη υπηρεσία τους (κόστος Σύνδεσης) συν το συνολικό κόστος μετακίνησης των υπηρεσιών (κόστος Μετακίνησης) για όλα τα στάδια. Το πρόβλημα Ανακατονομής K- Υπηρεσιών παρουσιάστηκε για πρώτη φορά από τους de Keijzer και Wojtczak [35], οι οποίοι μελέτησαν κυρίως την ειδική περίπτωση της μιας υπηρεσίας ($K = 1$).

Το πρώτο μας αποτέλεσμα για το πρόβλημα Ανακατανομής -Υπηρεσιών είναι ένα κάτω φράγμα ίσο με στον λόγο ανταγωνιστικότητας των ντετερμινιστικών άμεσων αλγορίθμων. Για να αποδείξουμε το κάτω φράγμα, δείχνουμε ότι οποιοσδήποτε άμεσος ντετερμινιστικός αλγόριθμος για το πρόβλημα Ανακατανομής K-Υπηρεσιών με λόγο ανταγωνιστικότητας c μπορεί να μετατραπεί σε άμεσο ντετερμινιστικό με λόγο ανταγωνιστικότητας c για το πρόβλημα K-Εξυπηρετητών. Αυτό σημαίνει ότι αν υπήρχε αλγόριθμος με λόγο ανταγωνιστικότητας μικρότερο από K για το πρόβλημα Ανακατανομής K-Υπηρεσιών, τότε θα μπορούσαμε να τον μετατρέψουμε σε έναν άμεσο ντετερμινιστικό αλγόριθμο για το πρόβλημα K-Εξυπηρετητών με λόγο μικρότερο από K. Αυτό οδηγεί σε άτοπο, αφού είναι ήδη γνωστό ότι δεν μπορεί να υπάρξει άμεσος ντετερμινιστικός αλγόριθμος με λόγο ανταγωνιστικότητας μικρότερο από K για το πρόβλημα των K-Εξυπηρετητών [55].

Γνωρίζοντας ότι δεν μπορούμε να αποφύγουμε την εξάρτηση του λόγου ανταγωνιστικότητας από τον αριθμό των διαθέσιμων υπηρεσιών (K), στρέφουμε την προσοχή μας στον σχεδιασμό ενός αλγορίθμου με λόγο ανταγωνιστικότητας, ο οποίος δεν εξαρτάται από τον αριθμό των πελατών (n). Αυτό το καταφέρνουμε για την ειδική περίπτωση, όπου έχουμε ακριβώς 2 διαθέσιμες υπηρεσίες ($K = 2$). Το δεύτερο αποτέλεσμα μας αφορά αυτή την περίπτωση και είναι ένας ντετερμινιστικός άμεσος αλγόριθμος με λόγο ανταγωνιστικότητας ίσο με 63. Το πρώτο βήμα του αλγορίθμου μας είναι επηρεασμένο από τον αλγόριθμο Double Coverage [20], ο οποίος λύνει βέλτιστα το πρόβλημα των K-Εξυπηρετητών στην ευθεία γραμμή. Σε αυτό το βήμα εξασφαλίζουμε ότι κάποια Υπηρεσία θα βρίσκεται σε κάθε στάδιο κοντά στο σύνολο των πελατών. Για να προσδιορίσει τις τελικές θέσεις των υπηρεσιών, ο αλγόριθμος μας πραγματοποιεί ένα δεύτερο βήμα, στο οποίο λαμβάνει υπόψη του και το κόστος Σύνδεσης των πελατών και μετακινεί αναλόγως τις υπηρεσίες στις τελικές τους θέσεις για το συγκεκριμένο στάδιο.

Δυναμική Χωροθέτηση Υπηρεσιών

Ολοκληρώνουμε την μελέτη μας πάνω σε άμεσα προβλήματα Δυναμικής Συνάθροισης, εξετάζοντας την άμεση εκδοχή του προβλήματος Δυναμικής Χωροθέτησης Υπηρεσιών. Το πρόβλημα Δυναμικής Χωροθέτησης Υπηρεσιών γενικεύει το κλασικό πρόβλημα πρόβλημα Χωροθέτησης Υπηρεσιών [44, 62] με την έννοια ότι ο μετρικός χώρος μεταξύ μεταξύ των υπηρεσιών και των πελατών αλλάζει σε κάθε στάδιο του προβλήματος. Ο στόχος σε αυτή την δυναμικά χρονοεξαρτώμενη παραλλαγή χωροθέτησης υπηρεσιών είναι η βελτιστοποίηση του αντισταθμίσματος μεταξύ της κλασικής αντικειμενικής συνάρτησης και

της σταθερότητας της λύσης. Η κλασική αντικειμενική συνάρτηση περιλαμβάνει ένα κόστος Ανοίγματος υπηρεσιάς και ένα κόστος Σύνδεσης του πελάτη στην κοντινότερη του υπηρεσία. Όσο αφορά την σταθερότητα της λύσης, η μοντελοποίηση επιπλέον περιλαμβάνει ένα κόστος Εναλλαγής, το οποίο χρεώνεται για κάθε αλλαγή σύνδεσης ενός πελάτη σε υπηρεσία μεταξύ δύο διαδοχικών σταδίων.

Το πρόβλημα Δυναμικής Χωροθέτησης Υπηρεσίων μελετήθηκε αρχικά από τους [37], οι οποίοι σχεδίασαν πιθανοτικό αλγόριθμο με λόγο προσέγγισης ίσο με $O(\log nT)$, όπου n ο αριθμός των πελατών και T ο αριθμός των σταδίων. Στη συνέχεια, αυτό το αποτέλεσμα βελτιώθηκε δραματικά από τους [5], οι οποίοι σχεδίασαν πιθανοτικό αλγόριθμο με λόγο προσέγγισης 14. Σε αυτή την διδακτορική διατριβή παρουσιάζουμε την πρώτη συστηματική μελέτη σχετικά με την άμεση εκδοχή του προβλήματος Δυναμικής Χωροθέτησης Υπηρεσιών, όπου η διαφορά σε σχέση με την μη άμεση εκδοχή του προβλήματος είναι ότι οι μετρικοί χώροι μεταξύ πελατών και υπηρεσιών αποκαλύπτονται ένας ένας σε κάθε στάδιο.

Τα πρώτα αποτελέσματά μας για το πρόβλημα Δυναμικής Χωροθέτησης Υπηρεσιών είναι δύο κάτω φράγματα στον λόγο ανταγωνιστικότητας των άμεσων ντετερμινιστικών αλγορίθμων και των άμεσων πιθανοτικών αλγορίθμων. Για το κάτω φράγμα των ντετερμινιστικών αλγορίθμων κατασκευάζουμε ένα στιγμιότυπο, στο οποίο κάθε άμεσος ντετερμινιστικός αλγόριθμος είναι αναγκασμένος να πληρώσει μεγάλο κόστος Εναλλαγής. Διαλέγοντας κατάλληλα τις παραμέτρους του προβλήματος, αποδεικνύουμε ένα κάτω φράγμα της τάξης $\Omega(m)$, όπου m είναι ο αριθμός των υπηρεσιών. Με παρόμοια κατασκευή και χρησιμοποιώντας την Αρχή του Υαο, αποδεικνύουμε ένα κάτω φράγμα της τάξης του $\Omega(\log m)$ για τους πιθανοτικούς άμεσους αλγορίθμους.

Όσο αφορά το άνω φράγμα για το πρόβλημα Δυναμικής Χωροθέτησης Υπηρεσιών, σχεδιάζουμε έναν αλγόριθμο, ο οποίος πραγματοποιεί δύο βήματα σε κάθε στάδιο. Στο πρώτο βήμα υπολογίζει μια *κλασματική λύση* λύνοντας την *γραμμική χαλάρωση* του προβλήματος Δυναμικής Χωροθέτησης Υπηρεσίων, η οποία περιλαμβάνει επιπρόσθετα και έναν όρο κανονικοποίησης στην αντικειμενική συνάρτηση. Ο όρος αυτός εξασφαλίζει ότι δύο διαδοχικές λύσεις δεν θα απέχουν πολύ μεταξύ τους και έτσι το κόστος Εναλλαγής θα παραμένει φραγμένο από το $O(\log m)$ σε σχέση με την βέλτιστη λύση. Στο δεύτερο βήμα μετατρέπουμε την κλασματική λύση σε μια *ακέραια λύση* για το πρόβλημα Δυναμικής Χωροθέτησης Υπηρεσιών, επιβαρύνοντας επιπλέον με έναν αθροιστικό όρο $O(\log n)$. Συνολικά, πετυχαίνουμε την κατασκευή ενός πιθανοτικού αλγορίθμου με λόγο ανταγωνιστικότητας ίσο με $O(\log m + \log n)$ για το πρόβλημα Δυναμικής Χωροθέτησης Υπηρεσιών.

Αντιστοιχία Όρων

Μετάφραση	Αγγλικός Όρος
Άθροισμα	Sum
Ακέραια λύση	Integer solution
Άμεση Βελτιστοποίηση	Online Optimization
Άμεση Μάθηση	Online Learning
Άμεσο πρόβλημα	Online problem
Άμεσος αλγόριθμος	Online algorithm
Αμετάκλητος	Irrevocable
Αναγωγή	Reduction
Ανακατανομή K-Υπηρεσιών	K-Facility Reallocation
Ανταγωνιστική Ανάλυση	Competitive Analysis
Άντικειμενική συνάρτηση	Objective function
Αντίπαλος	Adversary
Άνω φράγμα	Upper bound
Αποδοτικότητα	Efficiency
Αρχή Yao	Yao's principle
Βέλτιστος	Optimal
Δυναμική Συνάθροιση	Dynamic Aggregation
Δυναμική Χωροθέτηση Υπηρεσιών	Dynamic Facility Location
Δυναμικός	Dynamic
Εγγυήσεις	Guarantees
Ελάχιστο	Minimum
Κάλυψη Συνόλου Ελάχιστου-Αθροίσματος	Min-Sum Set Cover
Κανονικοποίηση	Regularization
Κάτω φράγμα	Lower bound
K-Εξυπηρετητές	K-server
Κλασματική λύση	Fractional solution
Κόστος Ανοίγματος Υπηρεσίας	Facility opening cost
Κόστος Εναλλαγής	Switching Cost
Κόστος Συνάθροισης	Aggregation cost
Κόστος Μετακίνησης	Moving Cost
Κόστος Πρόσβασης	Access Cost
Κόστος Σύνδεσης	Connection cost
Κυρτή Βελτιστοποίηση	Convex Optimization
Λόγος ανταγωνιστικότητας	Competitive ratio
Μάθηση	Learning

Μέγιστο	Maximum
Μέθοδος Ενημέρωσης Πολλαπλασιαστικών Βαρών	Multiplicative Weights Update Method
Μέσος όρος	Average
Μετάθεση	Permutation
Ντετερμινιστικός	Deterministic
Πελάτης	Client
Πιθανοτικός	Randomized
Προσπέλαση Λίστας	List Update
Στάδιο	Stage
Στατικός	Static
Υπηρεσία	Facility
Υπολογισμός Μέσου Όρου Αντιπάλων	Averaging over adversaries
Υποσύνολο	Subset

Contents

List of Figures

1 Introduction

Aggregation is the task of combining a set of objects in such a way that we can refer to them collectively as an aggregated object. The aggregated object is the result of an *Aggregate Function* and serves as the representative of the objects with respect to the Aggregate Function. Common aggregate functions include the Average, Sum and the Minimum. If we further hypothesize that the task of aggregation is performed in stages and at each stage we have to aggregate a different set of objects, then we have the concept of *Dynamic Aggregation. Dynamic Aggregation* problems arise in many applications such as epidemiology, vaccination planning, anti-virus design, management of human resources, urban planning, advertising and search engines.

A very topical example from epidemiology, which reveals the significance of Dynamic Aggregation, is monitoring the spread of a pandemic such as COVID-19 in a given population. The patterns of spread and the measures taken to prevent it vary dynamically across different populations and have temporal aspects depending on the weather, the mobility of people, their culture and many other factors. Therefore, it is necessary to conduct many aggregation tasks such as recognizing groups of people susceptible to conduct the virus, finding groups of people which could probably have a severe outcome due to the virus and many others. This groups may change abruptly due to new scientific data, mutations of the virus, weather conditions or behavioral transitions of the people. Another example from urban planning is the problem of aggregating (or clustering) in urban traffic networks. Traffic is a strongly time-variant process that needs to be studied in the spatiotemporal dimension in order to better understand and reveal the hidden information during the process of congestion formation and dissolution.

Advertising companies can also benefit from Dynamic Aggregation by recommending alternatives to users dynamically based on the users' temporal preferences. In a realistic scenario, users' preferences gradually change and the suggestions should adjust to the new information in order to increase the effectiveness of the recommendation. In this example, there is a limited number of alternatives, which are represented as an ordered list; alternatives which are closer to the front of the list are the most popular alternatives. A closely related instance of Dynamic Aggregation is that of a web search engine, such as Google. Each query asked might have many different meanings depending on the user. For example, the query "Python" might refer to an animal, a programming language or a movie. Given

the pages related to “Python”, a goal of the search engine algorithm is to rank them such that for each user, the pages of interest appear as high as possible in the ranking.

The aforementioned practical examples are instances of Dynamic Aggregation problems, in which the problem input is usually not known at the start and is revealed piece-by-piece. Algorithms that address such kind of problems are called *online algorithms*. In this thesis, we study the design of *online algorithms* for Dynamic Aggregation problems. An online algorithm can process its input piece-by-piece in the order that the input is fed to the algorithm, without having the entire input available from the start. Thus, online algorithms operate in a serial fashion and make decisions for a piece of the input before the next piece of the input arrives. We evaluate the performance of online algorithms by using the notion of the *competitive ratio*, which is the worst case ratio between the solution cost of an online algorithm and the solution cost of an optimal offline algorithm that knows the entire input (notions regarding online algorithms are defined in Section 1.3). The concept of online algorithms was introduced to capture more realistic scenarios where the whole data are not available before the execution of the algorithm, which is the case for offline algorithms.

Our main motivation is to design online algorithms for Dynamic Aggregation problems, which are competitive against the optimal offline solution. We study three remarkable online Dynamic Aggregation problems, namely the *Online Dynamic Facility Location problem* [37], the *Online K-Facility Reallocation problem* [35] and the *Online Min-Sum Set Cover problem* [18]. For these problems, we also investigate the limitations of online solutions by designing instances, where any online algorithm is forced to take suboptimal decisions due to lack of information about future requests. Our contribution is twofold in the sense that we improve the performance of the best known online algorithm and we prove lower bounds on the performance of any online algorithm for each problem. In order to achieve these results, we carefully craft techniques from the fields of competitive analysis, convex optimization and online learning and show interesting connections between these fields.

The introduction is structured as follows. In Section 1.1, we survey the results regarding Dynamic Aggregation problems and explore related areas. Then, in Section 1.2, we introduce the specific online Dynamic Aggregation problems studied in this thesis and present our results. Section 1.2 is subdivived into three subsections corresponding to the three online Dynamic Problems that we study. Finally, in Section 1.3, we discuss some basic concepts and notions regarding the scientific fields related to our work.

1.1 Dynamic Aggregation Problems

Aggregation problems have been extensively studied in the literature. The input to an Aggregation problem can be a set, a multiset, or a list from some input domain I and the respective aggregate function outputs an element of an output domain O. Typical examples

of aggregate functions include:

1. The Sum function, which is the addition of elements of mathematical objects such as numbers, functions, arrays, matrices or even polynomials.
2. The Average function, which is the sum of the numbers divided by how many numbers are being averaged.
3. The Count function, which determines the number of elements of a finite set of objects.
4. The Minimum and Maximum functions, which calculate the smallest and the largest value of the function respectively. The values are either calculated within a given range resulting to a local extreme, or on the entire domain resulting to a global extreme.

Dynamic Aggregation problems introduce an additional temporal dimension in Aggregation problems by letting the problems process in stages (stages can also be referred as rounds or timesteps). Thus, an algorithm for a Dynamic Aggregation problem solves a possible different instance of an Aggregation problem at each stage. This fact has a major impact on the objective function of a Dynamic Aggregation problem, which is extended to favor solutions that do not change dramatically between timesteps. The extension involves an additional fixed amount of a *switching cost* (or *moving cost*), which is incurred every time a solution between two consecutive time steps changes. Consequently, algorithms for Dynamic Aggregation problems aim at discovering temporal evolution of elements that is not too sensitive to transient changes.

Towards a more formal generic definition of Dynamic Aggregation problems, let T be the number of stages and let $f : I \; ! \; O$ be an aggregate function and $g : I^{0} \; ! \; O^{0}$ be a switching cost function. At each stage $t, 1 \quad t \quad T$, there is a request r_t having the form of a set, multiset, vector, matrix or list, which the algorithm has to serve. The objective function of the problem has the form:

$$\overset{T}{\underset{t=1}{\text{å}}} f(Sol_t) + \overset{T}{\underset{t=1}{\text{å}}} g(Sol_t, Sol_{t+1})$$

where the first term is the cost incurred by f at stage t due to to the solution Sol_t produced by the algorithm to serve the request r_t at stage t, and the second term is the switching cost incurred by g due to the distance between two consecutive solutions of the algorithm.

The family of the Dynamic Aggregation problems is closely related to another famous family of problems called Metrical Task Systems (MTS). In MTS, we are given a set of N states and a metric function d specifying the cost of moving between the states. At each step, a task arrives; the cost of serving the task at state i is c_i. An algorithm has to choose a state to

process the task. If it switches from state i to state j and processes the task there, it incurs a cost $d(i,j) + c_j$. Given an initial state and a sequence of requests, the goal is to process all tasks at minimum cost.

The most notable difference between Metrical Task Systems and Dynamic Aggregation problems is that MTS are inherently online, whereas Dynamic Aggregation problems are defined in both the offline and the online setting. Moreover, in Dynamic Aggregation problems the cost of serving a request is an aggregate function, while MTS allows more general functions to be considered. Additionally, the switching cost functions in Dynamic Aggregation problems usually measure the distance between two consecutive solutions provided by the algorithm as opposed to the switching cost of MTS, which measures the distance between two states (or configurations) of the algorithm. Despite the aforementioned minor differences, the cost of serving a task in MTS can be an aggregate cost function as well as to different states in MTS may correspond to two different solutions, thus almost all online problems considered in this thesis belong to both families.

The list update problem

A typical Dynamic Aggregation problem is the list update problem, where we are given a set of n items in a list and the cost of accessing an item is proportional to its distance from the first item in the list. An algorithm has to reorder the list by transposing elements so that the total cost of accesses is minimized. The reordering actions include two types transpositions:

- A free transposition of the accessed item.
- A paid transposition of unit cost for exchanging two adjacent items.

The objective of the list update problem can be written in the form

$$\sum_{t=1}^{T} f(Sol_t) + \sum_{t=1}^{T} g(Sol_t, Sol_{t+1})$$

by plugin in an aggregate function f that counts the position of the accessed item in the solution (list) at stage t and by counting paid transpositions between any two consecutive solutions with the Kendall tau distance, which counts the number of inversions between the two solutions. The list update problems is the special case of the online Min-Sum Set Cover, where the requests have the form of sets of items S_t and the access cost is the positions of the first element of S_t in the list. Moreover, all transpositions in online MSSC are paid.

The list update problem is one of the most classic and famous online problems [23], which has been studied comprehensively and extensively in the literature. The deterministic competitive ratio of the list update problem is at least $2 - \frac{2}{n+1}$ and there are several deterministic

algorithms achieving almost matching upper bounds. The most famous of them is the simple and intuitive Move-to-Front (MTF) algorithm, which moves the (unique) element of S_t to the first position of the permutation. MTF is known to be 2-competitive [66] and several other 2-competitive algorithms can be found in [1, 38]. Another intuitive algorithm for the list update problem is the Frequency Count (FC) algorithm, which orders the elements in decreasing order according to their frequencies. Despite the fact that the FC algorithm appears to be a suitable algorithm for the problem, it is surprising that it's competitive ratio is $\Omega(n)$. For randomized algorithms, the best known competitive ratio is 1.6 [2] and the best lower bound is 1.50115 [3].

Interestingly, all prior work on the list update problem does not seem to provide us with the right tools for obtaining an algorithm for online Min-Sum Set Cover, which we study in this thesis. Almost all natural generalizations of successful list update algorithms (e.g., Move-to-Front, Frequency Count) end up with a competitive ratio way far from the desired bound. In fact, even for sets with two elements, most of them have a competitive ratio depending on n, such as $\Omega(\sqrt{n})$ or even $\Omega(n)$.

The K-server problem

Another important and well-studied online optimization problem which falls into the category of Dynamic Aggregation problems is the K-server problem. In the K-server problem, we have K mobile servers located at some points of a metric space. The input is a request sequence $r = (r_1, \ldots r_T)$, where r_t is the point requested at time t in an online fashion. The goal is to minimize the total distance traveled by the servers for serving r:

$$\sum_{t=1}^{T} d(S_t, S_{t-1})$$

where S_t is the configuration of the algorithm at time t, which describes the positions of the K servers in the metric space at stage T. Observe that the K-server problem can be described as a Dynamic Aggregation problem, where the aggregated cost at stage t is the distance traveled by the server that serves r_t and the switching cost is the sum of all distances traveled by the other servers between two stages.

The K-server problem was introduced by Manasse et al. [60] as a far-reaching generalization of various online problems, the most notable of which is the paging problem. The paging problem is the special case of the k-server problem in a uniform metric, i.e. when all distances between distinct points are 1. Here, the K-servers correspond to the K slots in the cache, and the pages correspond to the points. Evicting a page from the cache and bringing a new one maps to moving a server between the corresponding points at a cost of 1. Manasse et al. [60] showed that the competitive ratio of deterministic algorithms is at

least k, even if the metric space contains only $n = K + 1$ points. For the paging problem, Sleator and Tarjan [66] showed that many natural algorithms are K-competitive.

The initial research for the K-server problem focused on special metrics like weighted stars, lines and trees, and for many cases tight K-competitive algorithms were obtained [30, 31] . For general metric spaces, Fiat et al. [41] obtained the first $f(k)$-competitive algorithm, with competitive ratio $O((k!)^3)$. Several improvements followed (but with ratio still exponential in K) until Koutsoupias and Papadimitriou [55] showed that the Work Function Algorithm (WFA) is $(2K \quad 1)$-competitive for every metric space. This impressive result remains up to date the best known upper bound on the deterministic competitive ratio of the K-server problem.

1.2 Considered Problems and Results

In this section, we present the online Dynamic Aggregation problems considered in this thesis, namely the Online Min-Sum Set Cover problem, the Online 2-Facility Reallocation Problem and the Online Dynamic Facility Location problem. The section is divided into three subsections, one for each considered online problem. We start with the Online Min-Sum Set Cover problem.

1.2.1 The Online Min-Sum Set Cover Problem

A typical representative of the class of online Dynamic Aggregation problems is the online Min-Sum Set Cover (MSSC). In MSSC, we are given a universe U on n elements and at each stage t, a subset $S_t \quad U$ arrives in an online fashion. The goal is to construct online T permutations $p_1, \ldots, p_T$ of elements of U so as to provide the best covering time for the subsets of S. The covering time of S_t is measured with the aggregate function $p_t(S_t)$ and is the position of the first element of S_t in p_t, i.e., $p_t(S_t) = \min\{i \mid p_t(i) \ 2 \ S_t\}$. The switching cost between two consecutive solutions is the number of inversions between p_t and p_{t+1}, known as the Kendall tau distance $d_{\mathrm{KT}}(p_t, p_{t+1})$. The goal is to minimize the total cost:

$$\sum_{t=1}^{T} \big(p_t(S_t) + d_{\mathrm{KT}}(p_t, p_{t+1})\big).$$

The motivating example for online MMSC is that of a web search engine, where each query asked might have many different meanings depending on the user and the goal of the search engine algorithm is to rank them such that for each user, the pages of interest appear as high as possible in the ranking. Another example news streams include articles covering different reader interests each. We want to rank the articles so that every reader finds an article of interest as high as possible. The MSSC problem serves as a theoretical model for practical problems of this type, where we want to aggregate disjunctive binary preferences

(expressed by the input sets) into a total order. E.g., for a news stream, the universe U corresponds to the available articles and the sets S_t correspond to different user types. The cost of a ranking (i.e., permutation on U) for a user type is the location of the first article of interest. Clearly, in such applications, users arrive online and the algorithm might need to re-rank the stream (i.e., change the permutation) based on user preferences.

The online MSSC generalizes the famous list update problem, where the sets have cardinality 1 ($|S_t| = 1$). However, all natural generalizations of nearly optimal algorithms for the list update problem fail to achieve a competitive ratio better than $\Omega(n)$, even if $|S_t| = 2$ for all t. This suggests that online MSSC has a distinctive combinatorial structure, very different from that of list update, whose algorithmic understanding calls for significant new insights. The main reason has to do with the disjunctive nature of the definition of the access cost $p(S_t)$. In list update, the optimal solution is bound to serve a request S_t by its unique element. The only question is how fast an online algorithm should upgrade it (and the answer is "as fast as possible"). In MSSC, the hard (and crucial) part behind the design of any competitive algorithm is how to ensure that the algorithm learns fast enough about the element e_t used by the optimal solution to serve each request S_t. This is evident in the highly adaptive nature of the deceptively simple greedy algorithm of [39] and in the adversarial request sequences for generalizations of Move-to-Front.

Related Work

The Min-Sum Set Cover problem has been mainly studied in the offline non dynamic version, where one seeks to find the best permutation p that covers all subsets $\{S_1, \ldots, S_m\}$ of S with minimum covering time. Since, the solution remains the same for all stages, there is no switching cost and therefore the goal is simply to minimize $\sum_t p(S_t)$. This variant of the MSSC problem generalizes various NP-hard problems such as Min-Sum Vertex Cover and Min-Sum Coloring and it is well-studied. Feige, Lovasz and Tetali [39] showed that the greedy algorithm, which picks in each position the element that covers the most uncovered sets, is a 4-approximation and that no $(4 - \varepsilon)$-approximation is possible, unless P = NP. Also, in [19] it was shown that greedy is no better than 4-approximation.

Multiple Intents re-ranking is an interesting generalization of MSSC, where for each set S_t, there is a *covering requirement* $K(S_t)$, and the cost of covering a set S_t is the position of the $K(S_t)$-th element of S_t in p. The DMSSC problem is the special case where $K(S_t) = 1$ for all sets S_t. Another notable special case is the Min-Latency Set Cover problem, which corresponds to the other extreme case where $K(S_t) = |S_t|$ [49]. Multiple Intents Re-ranking was first studied by Azar et. al. [11], who presented a $O(\log r)$-approximation; later $O(1)$-approximation algorithms were obtained [17, 65, 53]. Further generalizations have been considered, such as the Submodular Ranking problem, studied by Azar and Gamzu [10],

which generalizes both Set Cover and MSSC, and the Min-Latency Submodular Cover, studied by Im et.al [52].

It is easy to see that the online version of MSSC problem is a MTS, where the states correspond to permutations, thus $N = n!$, and the distance between two states is their Kendall tau distance. For a request set S_t, the request is a vector specifying the cost $p(S_t)$ for every permutation p. Although there has been a lot of work on understanding the structure of MTS problems [24, 34, 25, 55, 64, 63, 8, 14, 15], there is not a good grasp on how the structure relates to the hardness of MTS problems. Getting a better understanding on this area is a long-term goal, since it would lead to a systematic framework for solving online problems.

The online MSSC has an interesting connection with the prediction from expert advice problem. In this problem, there are N experts and expert i incurs a cost c_i^t in each step. A learning algorithm decides which expert i_t to follow (before the cost vector $\mathbf{c}^t$ is revealed) and incurs a cost of $c_{i_t}^t$. The landmark technique for solving this problem is the multiplicative weights update (MWU - a.k.a. Hedge) algorithm. For an in-depth treatment of MWU, we refer to [58, 46, 9].

In the classic online learning setting, there is no cost for moving probability mass between experts. However, in a breakthrough result, Blum and Burch [22] showed that MWU is $(1 + \epsilon)$-competitive against the best expert, even if there is a cost D for moving probability mass between experts. By adapting this result to MSSC (by viewing permutations as experts), we can get an (inefficient) randomized algorithm with competitive ratio $(1 + \epsilon)$, for any constant $\epsilon \in (0, 1/4)$.

Results

We consider the r-uniform case, where all request sets have the same size $|S_t| = r$ and initiate a systematic study on online MSSC.

The first of our main results is a tight bound on the deterministic competitive ratio of Online MSSC. This is achieved by showing first a lower bound of $(r + 1)(1 - \frac{r}{n+1})$ on the competitive ratio of deterministic algorithms. Note that for $r = 1$, this bound evaluates to $2 - \frac{2}{n+1}$, which is exactly the best known lower bound for the list update problem. Then, we complement this result by providing a matching (up to constant factors) upper bound of $(5r + 2)$ on the competitive ratio. Our algorithm uses a rounding scheme, to derandomize the multiplicative weights update (MWU) algorithm.

We then turn our attention to computational efficient algorithms and we propose a memoryless algorithm called *Move-All-Equally* (MAE). We show that MAE has a lower bound of $\Omega(r^2)$ and an upper bound of $2^{O(\sqrt{\log n \log r})}$ on its competitive ratio. and we conjecture that an $O(r)$ guarantee cannot be achieved by memoryless algorithms.

Finally, we study the much more general dynamic version of online MSSC, where the algorithm is compared against an optimal solution allowed to change permutations over time. We investigate the performance of the MAE algorithm and obtain an upper bound of $O(r^{3/2}\sqrt{n})$ on its competitive ratio. Although this guarantee is not very strong, we show that, rather surprisingly, it is essentially tight and no better guarantees can be shown for this algorithm by showing a lower bound of $\Omega(r\sqrt{n})$.

1.2.2 Online K-Facility Reallocation Problem

In the K-Facility Reallocation problem, K facilities are initially lying at points $(x_1^0, \ldots, x_K^0)$ of a metric space M. There are n clients, also residing on the same metric space, that use the facilities for T consecutive days. Each day, every client connects to the facility closest to their location and incurs a connection cost equal to this distance. Since the clients are free to move around on M from day to day, the algorithm can also move the facilities accordingly, to keep the connection cost low. Naturally, moving a facility is not free, but costs a price equal to the distance traversed. Our goal is to specify the exact positions of the facilities at each day so that the total connection cost plus the total moving cost over all T days is minimized:

$$\text{Cost}(x) = \sum_{t=1}^{T} \left[\sum_{k=1}^{K} |x_k^t - x_k^{t-1}| + \sum_{i=1}^{n} \min_{k \in K} |a_i^t - x_k^t| \right].$$

where a_i^t is the position of client i at stage t.

It is easy to see that the K-Facility Reallocation problem is a Dynamic Aggregation problem, where the aggregate function is the sum of distances of all clients to the facilities (connection cost) and the switching cost function is the distance travelled by the facilities at each stage. The Facility Reallocation problem was introduced by [35] and studied in the case the metric space is the real line. The motivating example considered in [35] for $K = 1$ consists of a political party moving along the spectrum from left to right wing, in an attempt to please more voters. Extending to K, this setting applies to clustering and advertising: following [57] from Yahoo Labs, companies often have a limited number of slots to suggest alternatives to users (such as ads or movie suggestions), given previously collected data. The users' preferences gradually change however and the limited number of suggestions need to stay enticing, without appearing to have abruptly adjusted to the new information.

For another intuitive example from the operations research field, consider a beach, where two ice cream vendors are to be located for the next three days. The beach is visited by ten customers for the next three days and these customers may change their location on the beach. Naturally, each customer wants to have an ice cream vendor close to him in order to buy ice cream. The goal is to minimize the total distance traveled by the customers

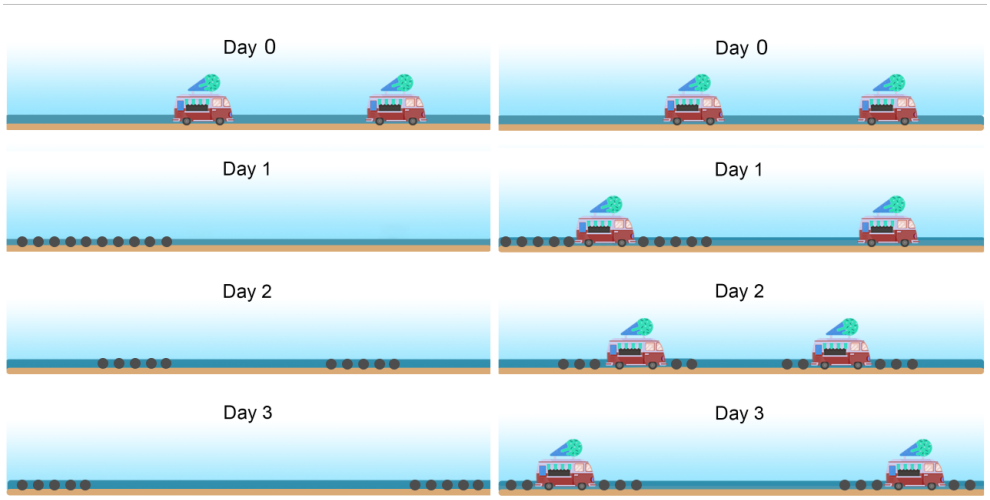


FIGURE 1.1: The picture on the left is an instance of 2-Facility Reallocation problem on the line. The initial ice cream vendor locations are illustrated in day 0 and the dots indicate the customer locations. The right picture shows a good solution for this instance, namely the facility locations at each day.

plus the total distance traveled by the ice cream vendors. Figure 1.1 depicts the instance on the left and the solution for this instance on the right respectively. The black dots are the customers, which appear in different locations throughout the days.

The previous example is an instance of the K-Facility Reallocation problem, where the number of mobile facilities is two (the ice cream vendors), the number of stages is three (the days) and the number of agents is 10 (the customers). Moreover, the metric space is the real line (the beach) and the variant is the offline if we know all customer locations throughout the days at the beginning of the first day. The problem is online if we learn the customer positions of the next day only after we have served (located the ice cream vendors) the customers of the current day.

Related Work

The K-Facility Reallocation problem was first studied by [35], who designed optimal offline and online algorithms for the case of $K = 1$ and presented a dynamic programming algorithm for $K \quad 1$ facilities with running time exponential in K. The result for general K in the offline variant was improved by [45], who presented an optimal algorithm with running time polynomial in the combinatorial parameters of K-Facility Reallocation (i.e., n, T and K). This substantially improves on the complexity of the algorithm, presented in [35], that is exponential in K. Their algorithm solves a Linear Programming relaxation and then *rounds* the *fractional solution* to determine the positions of the facilities. The main technical contribution is showing that a simple rounding scheme yields an integral solution that has the exact same cost as the *fractional one*.

The K-Facility Reallocation problem can be seen as multi-winner election (or committee selection) problem in utilitarian voting with single peaked preferences, especially under the Chamberlin-Courant rule. Two papers related to such problems are [7] and [61]. These deal with selecting the best among many possible outcomes in order to maximize the agents' utility. However, our setting is dynamic in the sense that the agents preferences change between stages, thus the goal is to minimize a social cost function over T stages and we also have to take into account that the solution provided at each stage should be close to the solution of the previous stage.

In [47], a mobile facility location problem was introduced, which can be seen as a one stage version of our problem. They showed that even this version of the problem is NP-hard in general metric spaces using an approximation preserving reduction from K-median problem.

Online facility location problems and variants have been extensively studied in the literature, see [44] for a survey. In [36], an online model where facilities can be moved with zero cost was studied. Despite achieving a constant competitive ratio, this model has the drawback that a misplaced facility can be moved for free to an optimal point without causing a penalty for the initial error. To remove this obstacle, [40] proposed a model, where moving a facility incurs a cost proportional to the distance it has moved.

The online variant of the K-Facility Reallocation problem is closely related to the K-server problem, which is one of the most natural online problems. [54] showed a $(2K \quad 1)$-competitive algorithm for the K-server problem for every metric space, which is also K-competitive, in case the metric is the real line [21]. Other variants of the K-server problem include the (H, K)-server problem [16, 13], the Infinite server problem [33] and the K-taxi problem[41, 32].

Regarding online clustering, [28] proposed an approach, which stems from Hierarchical Agglomerative Clustering. A set of K clusters is maintained while data points are presented in online fashion. Clusters can be *merged*, making space for an extra cluster to be used for incoming data. Clusters cannot be split however: this is both computationally expensive and would change the classification of preexisting data points, which is undesirable in hierarchical clustering. Also, the number of clusters is fixed from the start at K and it is impossible to 'buy' more of them. The fast increasing volume of available data and the requirement for responsive services has led to yet another approach by [57], namely *online* clustering algorithms balancing the quality of the clusters with their rate of change over time. The model is *semi*-online, meaning there is some information such as the length of the stream and the total clustering cost of optimal K-means. The algorithm proposed achieves a polylog competitive ratio using a polylogarithmicaly larger fraction of clusters. The authors also present a similar algorithm for the purely online case.

Results

Our first result is a lower bound of K on the deterministic competitive ratio for the online K-Facility Reallocation problem. In order to prove this lower bound, we show that no deterministic algorithm can achieve a better competitive ratio for the online K-Facility Reallocation problem that is better than the competitive ratio of the K-server problem. Since we know that the K-server problem has a lower bound of K, the same holds for K-Facility Reallocation.

The above arguments rule out the existence of a deterministic algorithm for the 2-Facility Reallocation problem with competitive ratio lower than 2. Thus, we focus on designing an online algorithm with competitive ratio that does not depend on the number of clients n. The result is a $O(1)$-competitive algorithm, which is inspired by the *double coverage algorithm* that solves optimally the K-server problem on the line.

1.2.3 The Online Dynamic Facility Location Problem

The last online Dynamic Aggregation problem we study in this thesis is the Online Dynamic Facility Location problem (ODFL). ODFL is the online variant of the Dynamic Facility Location problem, which was introduced by Eisenstat et al. [37] to model the temporal aspects of temporally evolving social or infrastructure networks. In this time-dependent variant of the Facility Location problem, clients or facilities may change their location over time and the goal is to achieve the best tradeoff between the optimal connections of clients to facilities and the stability of solutions between consecutive timesteps.

The temporal aspect of the Dynamic Facility Location problem is modeled by T metrics given on the same set of clients and facilities, each representing the metric at time round $t \in \{1, \ldots, T\}$. In the Online Dynamic Facility Location, the T metrics on clients and facilities are revealed one by one at each round. The online algorithm must make its decision before the metric of the next round is revealed and without knowing the total number of rounds. More formally:

In Online Dynamic Facility Location, we are given a set of facilities F, $|F| = m$, a set of clients C, $|C| = n$, a switching cost g and a facility opening cost f. At each round $t \in \{1, \ldots, T\}$, a new metric between clients and facilities is revealed with the form of a $n \times m$ dimensional vector d_t, which has entries corresponding to distances over $F \times C$. We denote by $d_t(i, j)$ the distance between client j and facility i at time t. At each round t, the goal is to find a subset $A_t \subseteq F$ of open facilities and an assignment $f_t : C \to A_t$ of clients

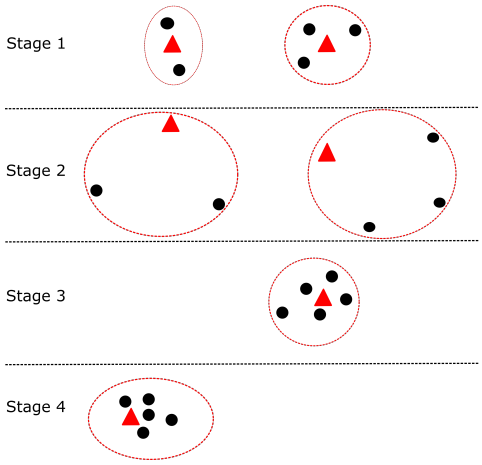


FIGURE 1.2: An instance of the Dynamic Facility Location problem, with $n = 5$ clients (the black circles) and $T = 4$ stages. At Stage 1, the algorithm decides to open two facilities (the red triangles) to serve the clients. At Stage 2, the clients move away from the facilities and increase the connection cost of the solution. At Stage 3, all clients are near and far from the previously two opened facilities. Thus, the algorithm decides to close the two facilities by paying the resulting switching cost and opens a new facility near the clients. At Stage 4, the clients have moved again away from the open facility, thus the algorithm closes the facility, pays the switching cost and opens a new facility near them.

to open facilities so as to minimize the objective:

$$f \sum_{t=1}^{T} jA_t j + \sum_{t=1, j2C}^{T} d_t(f_t(j), j) + g \sum_{t=1, j2C}^{T} \mathbb{1}\{f_t(j) \neq f_{t-1}(j)\}$$

where $\mathbb{1}\{p\}$ is the indicator function of proposition p. The assignment f_t of round t is chosen without knowing the distance vectors $d_{t+1}, \ldots, d_T$ of upcoming rounds. The objective function is the sum of the hourly opening costs for each open facility plus the connection costs of each client plus the switching costs (g per change of facility per client). We remark that any solution pays switching cost gn at round $t = 1$, since it switches to an initial assignment of the clients to facilities.

Observe that Online Dynamic Facility Location is a Dynamic Aggregation problem, where at each stage t the aggregate function are the sum of the first two term of the objective function and the switching cost function is the third term. In Figure 1.2, we illustrate an example of Dynamic Facility location with $n = 5$ clients and $T = 4$ different metrics.

Related Work

The offline and online variant of Facility Location have been studied extensively in the literature. For the offline Facility Location problem the approximability is $\Omega(\log n)$ [51] for the non-metric case while for the metric case the best lower bound is 1.463 [48] and the best algorithm has approximation ratio 1.488 [56]. Online Facility Location is known to have a competitive ratio of $\Omega(\log n / \log\log n)$ in the adversarial case for both deterministic and randomized algorithms [43] and constant competitive ratio if the clients are drawn from a known distribution [6].

In the Incremental Facility Location problem, which was first studied by [42], additional facilities can be opened and pairs of facilities can be merged at any point. The number of facilities does not need to remain fixed throughout and the final cost paid depends only on the number of open facilities *at the end*. Using techniques from streaming algorithms, the authors presented a constant competitive algorithm for incremental facility location as well as Incremental K-median clustering (also studied by [29]) using $O(K)$ additional clusters.

The study of Dynamic Facility Location so far concerns the offline case where the changes between distances of clients and facilities are known in advance. Eisenstat et al. [37] showed an upper bound of $O(\log nT)$ for the most interesting variant of Dynamic Facility Location with hourly facility costs, where facilities can be closed and are paid for all rounds in which they remain open. This result was later improved dramatically by [4], which gave an $O(1)$-approximation algorithm by exploiting a very interesting randomized rounding procedure.

Another problem closely related to Dynamic Facility Location, which is studied in the offline variant is the Dynamic Sum-Radii clustering problem, where clients arrive and depart, and the solution must be updated efficiently while remaining competitive with respect to the current optimal solution. In [50], they presented a data structure that maintains a solution whose cost is within a constant factor of the cost of an optimal solution in metric spaces with bounded doubling dimension an with worst-case update time logarithmic in the parameters of the problem.

Results

We give a comprehensive study on the competitive ratio of the Online Dynamic Facility Location problem. Towards this end, we provide lower bounds on the competitive ratio of deterministic and randomized algorithms and prove an almost matching upper bound on the competitive ratio of randomized algorithms.

Our first result considering ODFL is a lower bound of $\Omega(m)$ on the competitive ratio of any deterministic online algorithm. By using Yao's principle, we extend this result to prove that no randomized algorithm can achieve a competitive ratio better than $\Omega(\log m)$.

Regarding the upper bound, we design a randomized algorithm, which is $O(\log m + \log n)$-competitive. At each stage, the algorithm solves the LP relaxation of Dynamic Facility Location and then rounds the fractional solution to an integral one. We show that the fractional solution increases the switching cost by a factor of $O(\log m)$ and the rounding increases the facility cost by $O(\log n)$ resulting to an additive competitive ratio.

1.3 Technical Background

In this chapter, we present the main concepts and tools of online optimization problems. Online optimization problems where introduced to capture realistic scenarios, where the input of a problem is revealed piece-by-piece as opposed to traditional optimization problems, which assume complete knowledge of all data of a problem instance. Most of the times, we need to solve optimization problems, while taking decisions with incomplete information about the input. This observation has motivated the research on online optimization. An online algorithm processes its input sequentially in the order that is fed to the algorithm, without knowing the the pieces of input that will arrive in the future.

The field that studies optimization problems having no or incomplete knowledge of the future is called online optimization. Online optimization is challenging and it is not far from the truth that nearly all known algorithmic techniques have been applied in online optimization offer satisfactory solutions. Typically, the effectiveness of an online solution is measured with the method of *competitive analysis*.

1.3.1 Competitive Analysis

Competitive analysis, which was introduced by Sleator and Tarjan [66], is a method invented for analyzing online algorithms, which are forced to make decisions that may later turn out not to be optimal, since they have no access to the whole input. The study of online algorithms has focused on the quality of decision-making that is possible in this setting.

The performance of online algorithms is evaluated by comparing the quality of the produced solution against the solution from an optimal offline algorithm that knows the whole sequence of information a priori. Competitive analysis measures the performance of online algorithms using the notion of the competitive ratio. Since we focus on minimization problems in this thesis, we give the formal definition of the competitive ratio for minimization problems:

Definition 1.1. *Let $c > 1$ be a real number and let $ALG(s)$ be the cost of an online deterministic algorithm on a request sequence s. The algorithm is called c-competitive if there exists a constant b such that*

$$ALG(s) \quad c \; OPT(s) + b$$

holds for any request sequence s*, where* $OPT(s)$ *is the optimal offline algorithm which knows* s *in advance.*

Since a competitive algorithm should give guarantees that it performs well against any request sequence, we may assume that the request sequence is generated by a malicious adversary. The malicious adversary knows the strategy of the online algorithm and therefore can construct a request sequence which maximizes the ratio between the algorithm's cost and the optimal offline cost.

Observe that the competitive ratio of an online algorithm involves an additive constant b in its definition. The definition of competitiveness varies in the literature and mainly depends on the nature of the specific online problem. In some problems, the constant b may depend to instance specific parameters such as the the diameter of the metric space, in which the problem is studied. In some other problems, the goal is to achieve the best competitive ratio (the best constant c) assuming that $b = 0$. This is the notion of the *strict* competitiveness. In most cases, the additive constant b may depend on the initial configuration of the problem but it is required to not be dependent on the request sequence s.

Another important observation regarding the definition of the competitive ratio is that it does not involve a restriction on the computational resources that the online algorithm may need to produce a solution. The main principal for competitive analysis is the design of online algorithms that deal with uncertainties about the future in the best possible fashion. The competitive ratio is the loss factor that an online algorithm has to pay, since it has no access to future requests even though it may have unlimited computational power. However, in practical applications we may sacrifice the solution quality in favor of more computationally efficient solutions or seek for the best balance between solution quality and efficiency.

1.3.2 Randomized Algorithms

The need for online algorithms that achieve the best possible competitive ratio has led to the use of randomized algorithms. A randomized algorithm employs a degree of randomness as part of its strategy, typically using random bits as an auxiliary input to guide its behavior.

Randomized algorithms usually outperform significantly deterministic algorithms in terms of solution quality. In this case the competitive ratio depends on the type of the adversary with which the online randomized algorithm is compared. The most common adversary models are the following.

Oblivious Adversary: The oblivious adversary knows the code of the online algorithm, but it has no access to the random bits of the online algorithm. Furthermore, it has to construct the whole input before the algorithm begins its execution. Due to this limitations,

the oblivious adversary is sometimes called the *weak* adversary. Usually, randomization can be very helpful for improving the solution quality of an online algorithm against an oblivious adversary.

A randomized online algorithm ALG is c-competitive against oblivious adversaries if there exists a constant c, such that for any request sequence generated by an oblivious adversary,

$$\mathbf{E}[\mathrm{ALG}(s)] \quad c\,\mathrm{OPT}(s) + b.$$

Adaptive Online Adversary: The adaptive online adversary knows all actions of the algorithm, including its random choices. This adversary must make its own decision before it is allowed to know the decision of the algorithm and therefore is called the *medium* adversary. Specifically, at each step, the adversary gives a request maximizing the cost of the online algorithm. However, the adversary must also serve the each request in an online fashion. A randomized online algorithm ALG is c-competitive against adaptive online adversaries, if there exists a constant c, such that for any request sequence generated by an adaptive online adversary ADV,

$$\mathbf{E}[\mathrm{ALG}(s)] \quad c \quad \mathbf{E}[\mathrm{ADV}(s)] + b,$$

where $\mathrm{ADV}(s)$ is the cost of ADV to serve s.

There is another type of adversary for randomized algorithms, which is stronger than the oblivious adversary and the adaptive online adversary, called the *adaptive offline adversary*.

Adaptive Offline Adversary The adaptive offline adversary defers serving the request sequence until he has generated the last request. He then uses an optimal offline algorithm to serve it. In contrast, the adaptive online adversary must serve the input sequence online.

This adversary knows everything, even the random number generator, and is so strong that randomization does not help against it. Therefore it is sometimes called the *strong* adversary.

2 Methodologies and Techniques

In this chapter, we discuss the main methodologies and techniques we have used to provide our results for the online problems considered. We start with the potential function method.

2.1 Potential Function Method

We use the potential function method to analyze the performance of the MAE algorithm (Chapter 3) for the online Min-Sum Set Cover problem as well as to analyze the performance of our online algorithm for the K-Facility Reallocation problem on the line (Chapter 4). This method is extensively applied in computational complexity theory, in order to analyze the amortized time and space complexity of a data structure. Amortized analysis measures operations that smooth out the cost of infrequent but expensive operations.

Potential functions are an important tool for online optimization since they can be used for proving competitiveness results. We now summarize this technique for an online algorithm that attempts to minimize the cost of serving a request sequence $r = (r_1, \ldots, r_T)$ over T stages.

Let ALG be an online algorithm for some online minimization problem and OPT be the optimal offline algorithm for this problem. The potential Φ maps the current configurations of ALG and OPT to a non-negative value Φ. We denote by Φ the potential after request i. Let $\mathsf{ALG}(r_i)$ be the cost incurred by ALG on request r_i and $\mathsf{OPT}(r_i)$ be the cost incurred by the optimal offline algorithm on request r_i. The amortized cost a_i for serving request r_i is defined as

$$a_i = \mathsf{ALG}(r_i) + j\Phi_i \quad \Phi_{i-1}j$$

The intuition behind a potential function and the amortized cost is to measure how good the current configuration of the online algorithm is compared to an optimal offline algorithm. The potential can be viewed as a bank account. If the difference $j\Phi_i \quad \Phi_{i-1}j$ is negative, then the amortized cost underestimates the real cost $cost_i$. The difference is covered by withdrawal from the account. We have that the total cost of the online algorithm ALG for

serving request r is:

$$\sum_{i=1}^{T} \mathrm{ALG}(r_i) = \sum_{i=1}^{T} a_i + (\Phi_0 - \Phi_n) \leq \sum_{i=1}^{T} a_i + \Phi_0$$

where the second equality follows from a telescopic sum. Notice that Φ_0 is a constant that only depends on the initial configurations of ALG and OPT. Thus, up to an additive constant, the real cost is bounded from above by the amortized cost. If we can show that $a_i \leq c \cdot \mathrm{OPT}(r_i)$, where $\mathrm{OPT}(r_i)$ is the cost incurred by the optimal offline algorithm on request r_i, then it follows that the online algorithm ALG is c-competitive.

2.2 Averaging over Adversaries

In this section, we present the averaging technique for proving lower bounds in online optimization problems. This method has been applied in the K-server problem to provide a lower bound of K on the competitive ratio of deterministic algorithms [55] and in the list update problem to show that no deterministic algorithm can be better than $(2 - 2/(n+1))$-competitive, where n is the length of the list [23]. We use this method to provide a lower bound for the online Min-Sum Set Cover problem (Chapter 3).

Usually, the optimal offline cost is hard to bound both from above and below. For proving lower bound results on the competitive ratio of online algorithms, averaging over adversaries can help. The proof is based on a simple trick: Instead of comparing the online algorithm against one offline algorithm, we compare its cost against distinct offline algorithms. The basic idea is to have a set B of algorithms each of which serves the same request sequence r. If the sum of the costs of all algorithms in B is at most C, then there must be some algorithm in B which has cost at most $C/|B|$, the average cost. Hence, we get that $\mathrm{OPT}(r) \leq C/|B|$.

For example, the lower bound idea for the list update problem is constructed as follows: Given a deterministic algorithm ALG and a list with n elements, there is a cruel adversary, which chooses a sequence r of length T that always requests the last element in ALG's list. Hence, the online cost is the n for each request and $T \cdot n$ for serving the whole request sequence. The averaging technique considers all static offline algorithms that correspond to all permutations of the list. Each of these algorithms initially sorts the list according to its permutation and then keeps the list fixed for the rest of the sequence. The sorting cost of each algorithm can be bounded by a constant b that depends only on list size is negligible for large T. It is easy to prove that the average cost paid by the static algorithms is $b + T \cdot \frac{n+1}{2}$, thus for $T \to \infty$, the ratio between ALG and OPT becomes $\frac{2n}{n+1} = 2 - 2/(n+1)$.

2.3 Yao's Principle

In this section, we present Yao's principle (also called Yao's minimax principle or Yao's lemma) used to prove lower bounds on the competitive ratio of randomized algorithms. We apply Yao's principle in the online Dynamic Facility Location problem to show that the no randomized algorithm can be better than $\Omega(\log m)$-competitive, where m is the number of facilities (Chapter 5).

A lower bound on the competitive ratio is usually derived by providing a set of specific instances on which no online algorithm can perform well compared to an optimal offline algorithm. Here, again, we have to distinguish between deterministic and randomized algorithms. For deterministic algorithms, finding a suitable set of request sequences is in most cases comparatively easy. For randomized algorithms, however, it is usually very difficult to bound the expected cost of an arbitrary randomized algorithm on a specific instance from below. A standard tool for proving lower bounds for randomized algorithms is expressed below.

Theorem 2.1 (Yao's Principle for Online Problems)**.** *Let $fALG_y : y \in Yg$ denote the set of all deterministic online algorithms for an online minimization problem. If $\tilde{X}$ is a probability distribution over input sequences $fr_x : x \in Xg$ and $\tilde{c} > 1$ is a real number such that*

$$\inf_{y \in Y} \mathop{\mathbb{E}}_{\tilde{X}}[\mathrm{ALG}_y(r_x)] \quad \tilde{c} \mathop{\mathbb{E}}_{\tilde{X}}[\mathrm{OPT}(r_x)],$$

then $\tilde{c}$ is a lower bound on the competitive ratio of any randomized algorithm against an oblivious adversary.

Yao's principle states that the expected cost of a randomized algorithm on the worst-case input is no better than the expected cost for a worst-case probability distribution on the inputs of the deterministic algorithm that performs best against that distribution. Thus, it suffices to find an appropriate distribution of difficult inputs, and to prove that no deterministic algorithm can perform well against that distribution.

2.4 Multiplicative Weights Update Method

The multiplicative weights update method is an algorithmic technique most commonly used for decision making and prediction, and also widely deployed in game theory and algorithm design. We use one of the simplest use cases of this algorithm, called the Multiplicative Weights Update (MWU or Hedge) algorithm [58, 46, 9], to obtain an upper bound of $O(r)$ on the competitive ratio of the online Min-Sum Set Cover problem (Chapter 3). Our algorithm uses a rounding scheme to derandomize MWU.

Algorithm 1 Multiplicative Weights Update Algorithm

At every stage t, we have a weight w_i^t assigned to expert i. Initially, $w_i^1 = 1$ for all i. For each stage t, we associate the distribution $D^t = \{p_1, p_2, \ldots, p_n\}$ on the experts where $p_i^t = w_i^t / \sum_k w_k^t$. At stage t, we pick an expert according to distribution D^t and use it to make our prediction. Based on the outcome $j_t \in P$ in stage t, at stage $t+1$, the weight of expert i is updated as follows for each i:

$$w_i^{t+1} = \begin{cases} w_i^t(1-\epsilon)^{M(i,j^t)} & \text{if } M(i,j^t) \geq 0 \\ w_i^t(1+\epsilon)^{M(i,j^t)} & \text{if } M(i,j^t) < 0 \end{cases}$$

The MWU algorithm comes from the online learning setting and is the standard tool used for prediction with expert advice. In prediction with expert advice, a decision maker needs to iteratively decide on an expert whose advice to follow. We have N experts and expert i incurs a cost c_i^t in each step. A learning algorithm decides which expert i_t to follow (before the cost vector $\mathbf{c}^t$ is revealed) and incurs a cost of $c_{i_t}^t$. In the first round, all experts' opinions have the same weight. The decision maker will make the first decision based on the majority of the experts' prediction. Then, in each successive round, the decision maker will repeatedly update the weight of each expert's opinion depending on the correctness of his prior predictions. The weights are reduced in case of poor performance, and increased otherwise.

To motivate the Multiplicative Weights Update algorithm, consider the naive strategy that, at each iteration, simply picks an expert at random. The expected penalty will be that of the average expert. Suppose now that a few experts clearly outperform their competitors. This is easy to spot as events unfold, and so it is sensible to reward them by increasing their probability of being picked in the next round. Intuitively, being in complete ignorance about the experts at the outset, we select them uniformly at random for advice. This maximum entropy starting rule reflects our ignorance. As we learn who the hot experts are and who the duds are, we lower the entropy to reflect our increased knowledge. The multiplicative weight update is our means of skewing the distribution.

Towards a formal description of the MWU algorithm, let P be the set of events/outcomes. We assume there is a matrix M such that $M(i,j)$ is the penalty that expert i pays when the outcome is $j \in P$. The algorithm is described in Algorithm 1.

2.5 Regularization

An important technique used in online convex optimization is regularization. We use regularization to design a randomized $O(\log m + \log n)$-competitive algorithm for the online Dynamic Facility Location problem (ODFL), where m is the number of facilities and n is

the number of clients. (Chapter 5). Our algorithm regularizes the objective function of the linear program formulation of ODFL by adding a smooth convex function to its standard objective function. Then, the algorithm greedily solves the new online problem and obtains good bounds. The goal of regularization is to stabilize the solution so as to avoid drastic shifts in the solution from round to round that will lead to large switching cost.

The idea of regularization is age-old. One can think of "learning" the underlying phenomenon from the scarce observed data is an ill-posed inverse problem: out of many possible hypotheses that explain the data, which one should we choose? To restore uniqueness and reinforce the choice of simple models, regularization is the method that comes to mind. Support Vector Machines and many other successful algorithms arise from these considerations.

On the surface, it is not obvious why regularization methods would have anything to do with online learning. Indeed, the game described above does not aim at reconstructing some hidden phenomenon, as in the batch learning case. However, it is becoming apparent that regularization is indeed very natural. Just as regularization presents a cure to overfitting in the batch setting, so does regularization allow the online algorithm to avoid being fooled by an adversary. Indeed, blindly following the best decision given the past data implies, in some cases, playing into adversary's hands. Regularization is a way to choose "safe" decision.

3 Algorithms for the Static and Dynamic Online Min-Sum Set Cover Problem

3.1 Introduction

In this chapter, we consider the r-uniform version of OMSSC, where each S_t has cardinality r. We obtain tight bounds on the competitive ratio of deterministic online algorithms for MSSC against a static adversary, that serves the entire sequence by a single permutation. First, we show a lower bound of $(r + 1)(1 \quad \frac{r}{n+1})$ on the competitive ratio.

Theorem 3.1. *Any deterministic online algorithm for the Online Min-Sum Set Cover problem has competitive ratio at least* $(r + 1)(1 \quad \frac{r}{n+1})$.

Note that for $r = 1$, this bound evaluates to $2 \quad \frac{2}{n+1}$, which is exactly the best known lower bound for the list update problem. Since there are several algorithms matching this bound, this lower bound is the best possible for general values of r.

We complement this result by providing a matching (up to constant factors) upper bound.

Theorem 3.2. *There exists a* $(5r + 2)$*-competitive deterministic online algorithm for the Online Min-Sum Set Cover problem.*

Interestingly, all prior work on the list update problem (case $r = 1$) does not seem to provide us with the right tools for obtaining an algorithm with such guarantees! As we discuss in Section 3.2, virtually all natural generalizations of successful list update algorithms (e.g., Move-to-Front, Frequency Count) end up with a competitive ratio way far from the desired bound. In fact, even for $r = 2$, most of them have a competitive ratio depending on n, such as $\mathsf{W}(\sqrt{n})$ or even $\mathsf{W}(n)$.

This suggests that online MSSC has a distinctive combinatorial structure, very different from that of list update, whose algorithmic understanding calls for significant new insights. The main reason has to do with the disjunctive nature of the definition of the access cost $p(S_t)$. In list update, where $r = 1$, the optimal solution is bound to serve a request S_t by its unique element. The only question is how fast an online algorithm should upgrade it

(and the answer is "as fast as possible"). In MSSC, the hard (and crucial) part behind the design of any competitive algorithm is how to ensure that the algorithm learns fast enough about the element e_t used by the optimal solution to serve each request S_t. This is evident in the highly adaptive nature of the deceptively simple greedy algorithm of [39] and in the adversarial request sequences for generalizations of Move-to-Front, in Section 3.2.

To obtain the asymptotically optimal ratio of Theorem 3.2, we develop a rounding scheme and use it to derandomize the multiplicative weights update (MWU) algorithm. Our analysis bounds the algorithm's access cost in terms of the optimal cost, but it does not account for the algorithm's moving cost. We then refine our approach, by performing lazy updates to the algorithm's permutation, and obtain a competitive algorithm for online MSSC.

We also observe that based on previous work of Blum and Burch [22], there exists a (computationally inefficient) randomized algorithm with competitive ratio $1 + e$, for any $e \in (0, 1/4)$. This implies that no lower bound is possible, if randomization is allowed, and gives a strong separation between deterministic and randomized algorithms.

Memoryless Algorithms. While the bounds of Theorems 3.1 and 3.2 are matching, our algorithm from Theorem 3.2 is computationally inefficient since it simulates the MWU algorithm, which in turn, maintains a probability distribution over all $n!$ permutations. This motivates the study of trade-offs between the competitive ratio and computational efficiency. To this end, we propose a memoryless algorithm, called *Move-All-Equally* (MAE), which moves all elements of set S_t towards the beginning of the permutation at the same speed until the first reaches the first position. This is inspired by the Double Coverage algorithm from k-server [30, 31]. We believe that MAE achieves the best guarantees among all memoryless algorithms. We show that this algorithm can not match the deterministic competitive ratio.

Theorem 3.3. *The competitive ratio of the Move-All-Equally algorithm is* $\Omega(r^2)$.

Based on Theorem 3.7, we conjecture that an $O(r)$ guarantee cannot be achieved by a memoryless algorithms. We leave as an open question whether MAE has a competitive ratio $f(r)$, or a dependence on n is necessary. To this end, we show that the competitive ratio of MAE is at most $2^{O(\sqrt{\log n \log r})}$ (see Section 3.4 for details).

Dynamic Min-Sum Set Cover. We also consider the dynamic version of online MSSC. Dynamic MSSC is much more general and the techniques developed for the static case do not seem adequately powerful. This is not surprising, since the MWU algorithm is designed to perform well against the best static solution and not against a dynamic solution trajectory. We investigate the performance of the MAE algorithm. First, we obtain an upper bound on its competitive ratio.

Theorem 3.4. *The competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is* $O(r^{3/2}\sqrt{n})$.

Although this guarantee is not very strong, we show that, rather surprisingly, it is essentially tight and no better guarantees can be shown for this algorithm.

Theorem 3.5. *For any* $r \geq 3$*, the competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is* $\Omega(r\sqrt{n})$.

This lower bound is based on a carefully crafted adversarial instance; this construction reveals the rich structure of this problem and suggests that more powerful generic techniques are required in order to achieve any $f(r)$ guarantees. In fact, we conjecture that the lower bound of Theorem 3.1 is the best possible (ignoring constant factors) even for the dynamic problem and that using a work-function based approach such a bound can be obtained.

3.2 Lower Bounds on the Deterministic Competitive Ratio

We start with a lower bound on the deterministic competitive ratio of online MSSC. For the proof, we employ an averaging argument, similar to those in lower bounds for list update and k-server [60, 66]. In each step, the adversary requests the last r elements in the algorithm's permutation. Hence, the algorithm's cost is at least $(n - r + 1)$. Using a counting argument, we show that for any fixed set S_t of size r and any $i \in [n - r + 1]$, the number of permutations p with access cost $p(S_t) = i$ is $\binom{n-i}{r-1} r!(n-r)!$. Summing up over all permutations and dividing by $n!$, we get that the average access cost for S_t is $\binom{n+1}{r+1}\frac{r!(n-r)!}{n!} = \frac{n+1}{r+1}$. Therefore, the cost of the optimal permutation is a most $\frac{(n+1)}{r+1}$, and the competitive ratio of the algorithm at least $\frac{(n-r+1)(r+1)}{n+1}$.

Theorem 3.1. *Any deterministic online algorithm for the Online Min-Sum Set Cover problem has competitive ratio at least* $(r+1)(1 - \frac{r}{n+1})$.

Proof. Let ALG be any online algorithm. The adversary creates a request sequence in which every request is composed by the r last elements of the current permutation of ALG. At each round t, ALG incurs an accessing cost of at least $n - (r-1)$. Thus for the whole request sequence of m requests, $\text{Cost}(\text{ALG}) \geq m(n - r + 1)$.

The non-trivial part of the proof is to estimate the cost of the optimal static permutation. We will count total cost of all $n!$ static permutations and use the average cost as an upper bound on the optimal cost. For any request set S_t, we intend to find the total cost of the $n!$ permutations for S_t. To do this, we will count the number permutations that have access cost of i, for every $1 \leq i \leq n - (r-1)$. For such counting, there are two things to consider.

First, in how many different ways we can choose the positions where the r elements of S_t are located and second how many different orderings on elements of S_t and of $U \setminus S_t$ exist. We address those two separately.

1. For a permutation p that incurs an access cost of i, it follows that, from the elements in S_t, the first one in p is located in position i and no other element from the set is located in positions $j < i$. The other $r - 1$ elements of S_t are located among the last $n - i$ positions of p. There are $\binom{n-i}{r-1}$ different ways to choose the locations of those elements.

2. Once the positions of elements of S_t have been fixed, there are $r!$ different ways to assign the elements in those positions, equal to the number of permutations on r elements. Similarly, there are $(n - r)!$ different ways to assign elements of $U \setminus S_t$ to the $n - r$ remaining positions.

Gathering the above, we conclude that the number of permutations that incur access cost exactly i for a fixed request S_t is

$$\binom{n-i}{r-1} r!(n-r)!.$$

The latter implies two basic facts:

1. $\sum_{i=1}^{n-r+1} \binom{n-i}{r-1} r!(n-r)! = n!$ (since each permutation has a specific cost for request S_t).

2. The total sum of access costs for fixed request of size r is:

$$\begin{aligned}
\text{Total-Access-Cost} &= \sum_{i=1}^{n-r+1} i \binom{n-i}{r-1} r!(n-r)! \\
&= \sum_{i=1}^{n-r+1} \sum_{j=i}^{n-r+1} \binom{n-j}{r-1} r!(n-r)! \quad \text{By reordering the terms} \\
&= \sum_{i=1}^{n-r+1} \underbrace{\sum_{j=i}^{n-r+1} \binom{n-j}{r-1} r!(n-r)!}_{\text{permutations with access cost } i} \\
&= \sum_{i=1}^{n-r+1} \binom{n-i+1}{r} r!(n-r)! \\
&= r!(n-r)! \binom{n+1}{r+1}
\end{aligned}$$

where the last equality follows by the fact that $\sum_{i=1}^{n-r+1} \binom{n-i}{r-1} r!(n-r)! = n!$ (see number 1 above) with $n \leftarrow n+1$ and $r \leftarrow r+1$. Hence for a request sequence of length m, we get that

$$\text{Cost(OPT)} \quad m \ \frac{r!(n \quad r)!}{n!}\binom{n+1}{r+1} = m \ \frac{n+1}{r+1}.$$

We conclude that for any deterministic algorithm ALG, we have:

$$\frac{\text{Cost(ALG)}}{\text{Cost(OPT)}} \quad \frac{m \ (n \quad r+1)}{m \ \frac{n+1}{r+1}} = (r+1) \ \left(1 \quad \frac{r}{n+1}\right) = r+1 \quad \frac{r(r+1)}{n+1}. \ \square$$

Lower Bounds for Generalizations of Move-to-Front. For list update, where $r = 1$, simple algorithms like Move-to-Front (MTF) and Frequency Count achieve an optimal competitive ratio. We next briefly describe several such generalizations of them and show that their competitive ratio depends on n, even for $r = 2$.

$\text{MTF}_{\text{first}}$: Move to the first position (of the algorithm's permutation) the element of S_t appearing first in p_t.
Lower bound: Let the request sequence $S_1, S_2, \ldots S_m$, in which S_t contains the last two elements of $\text{MTF}_{\text{first}}$' s permutation at round $t \quad 1$. Formally, $S_t = fp_t(n \quad 1), p_t(n)g$. $\text{MTF}_{\text{first}}$ moves the first element of the request in the first position and in every round the last element in $\text{MTF}_{\text{first}}$' s permutation remains the same ($p_t(n) = p_0(n)$). As a result, $\text{MTF}_{\text{first}}$ pays $\text{W}(n)$ in each request, whereas OPT has the element $p_0[n]$ in the first position and just pays 1 per request.

MTF_{last}: Move to the first position the element of S_t appearing last in p_t.
Lower bound: Let the request sequence $S_1, S_2, \ldots S_m$, in which each set S_t always contains the last element of $p_t \quad 1$ and the fixed element 1. Clearly MTF_{last} pays $\text{W}(m \quad n)$, while $cost(\text{OPT}) = m$ by having element 1 in the first place.

MTF_{all}: Move to the first r positions all elements of S_t (in the same order as in p_t) .
Lower bound: The same as previous.

$\text{MTF}_{\text{random}}$: Move to the first position an element of S_t selected uniformly at random.
Lower bound: Let the request sequence $S_1, S_2, \ldots S_m$, in which each set S_t always contains an element selected uniformly at random from p_t and the fixed element 1. Therefore elements in the last $n/2$ positions of p_t have probability $1/2$ to be chosen. At each round t, $\text{MTF}_{\text{random}}$ moves with probability $1/2$ to the first position of the list, the element of S_t that was randomly selected. Thus at each round t, $\text{MTF}_{\text{random}}$ pays with probability $1/4$, moving cost at least $n/2$, meaning that the overall expected cost is at least $m \quad n/8$. As a result, the ratio is $\text{W}(n)$ since OPT pays m by keeping element 1 in the first position.

MTF_{last}, MTF_{all} and $\text{MTF}_{\text{random}}$ have a competitive ratio of $\Omega(n)$ when each request S_t consists of a fixed element e (always the same) and the last element in p_t, because they all incur an (expected for $\text{MTF}_{\text{random}}$) moving cost of $\Omega(n)$ per request.

The algorithms seen so far fail for the opposite reasons: $\text{MTF}_{\text{first}}$ cares only about the first element and ignores completely the second, and the others are very aggressive on using the second (rth) element. A natural attempt to balance those two extremes is the following.

$\text{MTF}_{\text{relative}}$: Let i be the position of the first element of S_t in p_t. Move to the first positions of the algorithm's permutation (keeping their relative order) all elements of S_t appearing up to the position $c \cdot i$ in p_t, for some constant c.
Lower bound: Let the request sequence $S_1, \ldots S_n$ in which S_t contains the $\lfloor \frac{n}{c} - 1 \rfloor$th and the nth element of the list at round $t - 1$. $\text{MTF}_{\text{relative}}$ never moves the last element and thus $p_n(0)$ belongs in all sets S_t. As in first case, this provides an $\Omega(n)$ ratio.

All generalizations of MTF above are memoryless and they all fail to identify the element by which optimal serves S_t. The following algorithm tries to circumvent this by keeping memory and in particular the frequencies of reqested elements.

$\text{MTF}_{\text{count}}$: Move to the first position the most frequent element of S_t (i.e., the element of S_t appearing in most requested sets so far).
Lower bound: The request sequence $S_1, \ldots, S_m$ is specifically constructed so that $\text{MTF}_{\text{count}}$ never moves the last b elements of the initial permutation p_0.

$$p_0 = [\underbrace{x_1, \ldots, x_{n-b}}_{n-b \text{ elements}}, \underbrace{x_{n-b+1}, \ldots, x_n}_{b \text{ elements}}]$$

The constructed request sequence $S_1, \ldots, S_m$ will be composed by m/n sequences of length n. Each piece of length n will have the following form:

1. $n - b$ requests $\{x_1, x_2\}, \{x_1, x_3\}, \ldots \{x_1, x_{n-b}\}$ (all requests contain x_1).
2. {element in position $n - b, x_{n-b+i}\}$ for $i = 1$ to b (additional b requests).

After the requests of type 1, the list is the same as the initial one, since x_1 has frequency $n - b$ and $x_2, \ldots x_{n-b}$ have frequency 1. Now consider the requests of type 2. $\text{MTF}_{\text{count}}$ moves always to the front the element which is in position $n - b$, since has already been involved in a type 1 request and has greater frequency. Therefore, $\text{MTF}_{\text{count}}$ pays $b \cdot (n - b)$. Repeating the same request sequence m/n times, we can construct a sequence of length m. In this request sequence, OPT keeps the element x_1 in the first position and the elements $\{x_{n-b+1}, \ldots, x_n\}$ in the next b positions. Thus, OPT pays $(n - b) \cdot m/n$ for the requests of type 1 and $b^2 \cdot m/n$ for the requests of type 2. $\text{MTF}_{\text{count}}$ pays $(n - b) \cdot m/n$ for the requests of type 1 (same as OPT), but $(n - b) \cdot b \cdot m/n$ for the requests of type 2. Setting $b = \sqrt{n}$, we get a $\Omega(\sqrt{n})$ lower bound for the competitive ratio of $\text{MTF}_{\text{count}}$.

Concluding this section, we remark that although the MTF algorithm achieves the best possible competitive ratio against the optimal dynamic solution for $r = 1$, it's natural generalizations are far from closing the gap with the lower bound of theorem 3.1 even when $r = 2$ and against the optimal static solution. The reason for this phase transition is that if $r \quad 2$ all these algorithms fail to *get closer* to the optimal solution which is the basic idea of the MTF algorithm for $r = 1$. As a result, we have to turn our attention to complicated memory-keeping strategies such as the one Algorithm 3 realizes by running the MWU algorithm in the background.

3.3 An Algorithm with Asymptotically Optimal Competitive Ratio

Next, we present algorithm Lazy-Rounding (Algorithm 3) and analyze its competitive ratio. The following is the main result of this section:

Theorem 3.2. *There exists a* $(5r + 2)$*-competitive deterministic online algorithm for the Online Min-Sum Set Cover problem.*

The remainder of this section is devoted to the proof of Theorem 3.2. At a high-level, our approach is summarized by the following three steps:

1. We use as black-box the multiplicative weights update (MWU) algorithm with learning rate $1/n^3$. Using standard results from learning theory, we show that its expected access cost is within a factor $5/4$ of OPT, i.e., $\mathsf{AccessCost}(\mathsf{MWU}) \quad \frac{5}{4}\,\mathsf{Cost}(\mathsf{OPT})$ (Section 3.3.1).

2. We develop an online rounding scheme, which turns any randomized algorithm A into a deterministic one, denoted $\mathsf{Derand}(A)$, with access cost at most $2r \quad \mathsf{E}[\mathsf{AccessCost}(A)]$ (Section 3.3.2). However, our rounding scheme does not provide any immediate guarantee on the moving cost of $\mathsf{Derand}(A)$.

3. Lazy-Rounding is a lazy version of Derand(MWU) that updates its permutation only if MWU's distribution has changed a lot. A *phase* corresponds to a time interval that Lazy-Rounding does not change its permutation. We show that during a phase:

   (a) The upper bound on the access cost increases, compared to Derand(MWU), by a factor of at most 2, i.e., $\mathsf{AccessCost}(\text{Lazy-Rounding}) \quad 4r \quad \mathsf{E}[\mathsf{AccessCost}(\mathsf{MWU})]$ (Lemma 3.6).

(b) The (expected) access cost of MWU is at least n^2. Since our algorithm moves only once per phase, its movement cost is at most n^2. Thus we get that (Lemma 3.7):

$$\text{MovingCost(Lazy-Rounding)} \quad \text{E[AccessCost(MWU)]}.$$

For the upper bound on the moving cost above, we relate how much MWU's distribution changes during a phase, in terms of the total variation distance, to the cost of MWU and the cost of our algorithm.

Based on the above properties, we compare the access and the moving cost of Lazy-Rounding against the access cost of MWU and to get the desired competitive ratio:

$$\text{Cost(Lazy-Rounding)} \quad (4r+1)\,\text{E[AccessCost(MWU)]} \quad (5r+2)\,\text{Cost(OPT)}.$$

Throughout this section we denote by $d_{\text{TV}}(d, d')$ the total variation distance of two discrete probability distributions $d, d' : [N] \to [0,1]$, defined as $d_{\text{TV}}(d, d') = \sum_{i=1}^{N} \max\{0, d(i) - d'(i)\}$.

3.3.1 Using Multiplicative Weights Update in Online Min-Sum Set Cover

In this section, we explain how the well-known MWU algorithm [58, 46] is used in our context.

The MWU Algorithm. Given $n!$ permutations of elements of U, the algorithm has a parameter $\beta \in [0,1]$ and a weight w_p for each permutation $p \in [n!]$, initialized at 1. At each time step the algorithm chooses a permutation according to distribution $\mathrm{P}_p^t = w_p^t / (\sum_{p \in [n!]} w_p^t)$. When request S_t arrives, MWU incurs an expected access cost of

$$\text{E[AccessCost(MWU}(t))] = \sum_{p \in [n!]} \mathrm{P}_p^t \cdot p(S_t)$$

and updates its weights $w_p^{t+1} = w_p^t \cdot \beta^{p(S_t)}$, where $\beta = e^{-1/n^3}$; this is the so-called *learning rate* of our algorithm. Later on, we discuss the reasons behind choosing this value.

On the Access Cost of MWU. Using standard results from learning theory [58, 46] and adapting them to our setting, we get that the (expected) access cost of MWU is bounded by Cost(OPT). This is formally stated in Lemma 3.1, which states that the MWU is $5/4$-competitive for access costs.

Lemma 3.1. *For any request sequence* $s = (S_1, \ldots, S_m)$ *we have that*

$$\mathsf{E}[\mathsf{AccessCost}(\mathsf{MWU})] \quad \frac{5}{4}\ \mathsf{Cost}(\mathsf{OPT}) + 2n^4 \ln n\,.$$

Proof. By the standard results in learning theory [58, 46], we know that for any sequence $s = (S_1, \ldots, S_m)$, the MWU algorithm satisfies

$$\sum_{t=1}^{m} \sum_{p \in [n!]} \mathsf{P}_p^t\ \ \mathsf{AccessCost}(p, S_t) \quad \frac{\ln(1/b)}{1-b}\ \min_{p \in [n!]} \sum_{t=1}^{m} p(S_t) + \frac{\ln(n!)}{1-b}.$$

where $b = e^{-1/n^3}$. Thus, $3/4 < b < 1$, for any $n \geq 2$. Using standard inequalities we get that $\frac{\ln(1/b)}{1-b} \leq 5/4$ and $1 - b \leq 1/2n^3$ for any $n \geq 2$. We finally get that,

$$\mathsf{E}[\mathsf{AccessCost}(\mathsf{MWU})] \quad \frac{5}{4}\ \mathsf{Cost}(\mathsf{OPT}) + 2n^4 \ln n.$$

□

On the Distribution of MWU. We now relate the expected access cost of the MWU algorithm to the total variation distance among MWU's distributions. More precisely, we show that if the total variation distance between MWU's distributions at times t_1 and t_2 is large, then MWU has incurred a sufficiently large access cost. The proof of the following makes a careful use of MWU's properties.

Lemma 3.2. *Let* P^t *be the probability distribution of the MWU algorithm at time* t*. Then, the probability distribution* P^{t+1} *of the algorithm satisfies*

$$d_{\mathsf{TV}}(\mathsf{P}^t, \mathsf{P}^{t+1}) \quad \frac{1}{n^3}\ \mathsf{E}[\mathsf{AccessCost}(\mathsf{MWU}(t))].$$

Proof. To simplify notation, let $W^t = \sum_{p \in [n!]} w_p^t$. We remind that by the definition of MWU, $w_p^{t+1} = w_p^t\ e^{-p(S_t)/n^3}$. Moreover, by the definition of total variation distance,

$$d_{\mathsf{TV}}(\mathsf{P}^t, \mathsf{P}^{t+1}) = \sum_{p:\mathsf{P}_p^t > \mathsf{P}_p^{t+1}} \mathsf{P}_p^t\ \ \mathsf{P}_p^{t+1} = \sum_{p:\mathsf{P}_p^t > \mathsf{P}_p^{t+1}} \left(\frac{w_p^t}{W^t}\ \ \frac{w_p^{t+1}}{W^{t+1}}\right)$$
$$\sum_{p:\mathsf{P}_p^t > \mathsf{P}_p^{t+1}} \left(\frac{w_p^t}{W^t}\ \ \frac{w_p^{t+1}}{W^t}\right)$$

$$\sum_{p \in [n!]} \left(\frac{w_p^t}{W^t}\ \ \frac{w_p^{t+1}}{W^t}\right) = \sum_{p \in [n!]} \frac{w_p^t}{W^t}\ \left(1\ \ e^{-p(S_t)/n^3}\right)$$

$$= \sum_{p \in [n!]} P_p^t \left(1 - e^{-(p(S_t)/n^3)}\right) \sum_{p \in [n!]} P_p^t \frac{p(S_t)}{n^3}$$

$$= \frac{1}{n^3} \mathrm{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))].$$

In the first inequality we used that $W^{t+1} \geq W^t$. In the second inequality we used that for all p we have that $w_p^{t+1} \leq w_p^t$ which implies that $\frac{w_p^t - w_p^{t+1}}{W^t} \geq 0$. In the last inequality we used that $1 - e^{-x} \leq x$, for any x. □

The following lemma is useful for the analysis of Lazy-Rounding. Its proof follows from Lemma 3.2 and the the triangle inequality.

Lemma 3.3. *Let t_1 and t_2 two different time steps such that $d_{\mathrm{TV}}(P^{t_1}, P^{t_2}) \geq 1/n$. Then, during the time interval $[t_1, t_2)$ the cost of the* MWU *algorithm is at least n^2,*

$$\sum_{t=t_1}^{t_2-1} \mathrm{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))] \geq n^2.$$

Proof. By Lemma 3.2 and summing over all t such that $t_1 \leq t < t_2$, we have that

$$\sum_{t=t_1}^{t_2-1} d_{\mathrm{TV}}(P^t, P^{t+1}) \geq \frac{1}{n^3} \sum_{t=t_1}^{t_2-1} \mathrm{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))]. \tag{3.1}$$

By triangle inequality we have that $d_{\mathrm{TV}}(P^{t_1}, P^{t_2}) \leq \sum_{t=t_1}^{t_2-1} d_{\mathrm{TV}}(P^t, P^{t+1})$. Combined with (3.1), this implies that

$$d_{\mathrm{TV}}(P^{t_1}, P^{t_2}) \geq \frac{1}{n^3} \sum_{t=t_1}^{t_2-1} \mathrm{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))].$$

By rearranging and using that $d_{\mathrm{TV}}(P^{t_1}, P^{t_2}) \geq 1/n$, we get that

$$\sum_{t=t_1}^{t_2-1} \mathrm{E}[\mathrm{AccessCost}(\mathrm{MWU}(t))] \geq n^3 \cdot \frac{1}{n} = n^2$$ □

3.3.2 Rounding

Next, we present our rounding scheme. Given as input a probability distribution d over permutations, it outputs a fixed permutation r such that for each possible request set S of size r, the cost of r on S is within a $O(r)$ factor of the expected cost of the distribution d on S. For convenience, we assume that n/r is an integer. Otherwise, we use $\lceil n/r \rceil$.

Algorithm 2 Greedy-Rounding (derandomizing probability distributions over the permutations)

Input: A probability distribution d over $[n!]$.

Output: A permutation $r \in [n!]$.

1: $R \leftarrow U$
2: **for** $i = 1$ to n/r **do**
3: $\quad S^i \leftarrow \arg\min_{S \subseteq R, |S|=r} \mathbb{E}_{p \sim d}[p(S)]$
4: $\quad$ Place the elements of S^i (arbitrarily) from positions $(i-1)r+1$ to ir of r.
5: $\quad R \leftarrow R \setminus S^i$
6: **end for**
Return r

Our rounding algorithm is described in Algorithm 2. At each step, it finds the request S with minimum expected covering cost under the probability distribution d and places the elements of S as close to the beginning of the permutation as possible. Then, it removes those elements from set R and iterates. The main claim is that the resulting permutation has the following property: *any request S of size r has covering cost at most $O(r)$ times of its expected covering cost under the probability distribution d.*

Theorem 3.6. *Let d be a distribution over permutations and let r be the permutation output by Algorithm 2 on d. Then, for any set S, with $|S| = r$,*

$$r(S) \leq 2r \cdot \mathop{\mathbb{E}}_{p \sim d}[p(S)].$$

Proof. Let e be the element used by r to serve the request on set S. Pick k such that $(k-1)r+1 \leq \mathsf{AccessCost}(r,S) \leq kr$. That means, e was placed at the permutation r at the kth iteration of the rounding algorithm.

Let $S^1, \ldots, S^{n/r}$ be the sets chosen during the rounding algorithm. Recall that r uses an element from S^k to serve the request. To this end, we use the technical Lemma 3.4 in order to get a lower bound on the expected cost of d. We distinguish between two cases:

1. Case $S = S^k$. In that case, by Lemma 3.4 we get that $\mathbb{E}_{p \sim d}[p(S)] \geq \frac{k+1}{2}$.
2. Case $S \neq S^k$. That means, e is one element of S in S_k and no elements of S are in sets $S_1, \ldots, S_{k-1}$. By construction of or rounding algorithm, we have that

$$\mathop{\mathbb{E}}_{p \sim d}[p(S)] \geq \mathop{\mathbb{E}}_{p \sim d}[p(S_k)] \geq \frac{k+1}{2}.$$

We get that in both cases $\mathbb{E}_{p \sim d}[p(S)] \geq \frac{k+1}{2}$. We conclude that

$$\frac{\mathsf{AccessCost}(r, S)}{\mathsf{E}_{p \quad d}[\mathsf{AccessCost}(d, S)]} \quad \frac{k \quad r}{\frac{k+1}{2}} \quad 2 \quad r \quad \square$$

We now proceed to the lemma omitted in the proof of Theorem 3.6.

Lemma 3.4. *Let d be a probability distribution over permutations and $1 \quad k \quad \frac{n}{r}$. Let $S_1, \ldots, S_k$ be disjoint sets such that $S_j \quad U$ and $jS_jj = r$ for all $1 \quad j \quad k$. Let $E_j = \mathsf{E}_{p \quad d}[p(S_j)]$ for any $1 \quad j \quad k$. If $E_1 \quad \ldots \quad E_k$, then, we have that $E_j \quad \frac{j+1}{2}$, for $1 \quad j \quad k$.*

Proof. We have that

$$\begin{aligned}
E_j \quad \frac{1}{j} \sum_{`=1}^{j} E_` &= \frac{1}{j} \sum_{`=1}^{j} \sum_{p} \Pr_d[p] \quad p(S_`) && \text{Using } E_1 \quad \ldots \quad E_j \\
&= \frac{1}{j} \sum_{p \, 2 \, [n!]} \Pr_d[p] \quad \sum_{`=1}^{j} p(S_`) && \text{Linearity of summation} \\
&\quad \frac{1}{j} \sum_{p \, 2 \, [n!]} \Pr_d[p] \quad \frac{j(j+1)}{2} \\
&= \frac{j+1}{2} \sum_{p} \Pr_d[p] = \frac{j+1}{2}
\end{aligned}$$

where $\sum_{`=1}^{j} p(S_`) \quad \frac{j(j+1)}{2}$ follows by the fact that $p(S_`)$ take $j$ different positive integer values (the sets $S_`$ are disjoint). $\square$

3.3.3 The Lazy Rounding Algorithm

Lazy-Rounding, presented in Algorithm 3, is essentially a lazy derandomization of MWU. At each step, it calculates the distribution on permutations maintained by MWU. At the beginning of each *phase*, it sets its permutation to that given by Algorithm 2. Then, it sticks to the same permutation for as long as the total variation distance of MWU's distribution at the beginning of the phase to the current MWU distribution is at most $1/n$. As soon as the total variation distance exceeds $1/n$, Lazy-Rounding starts a new phase.

The main intuition behind the design of our algorithm is the following. In Section 3.3.2 we showed that Algorithm 2 results in a deterministic algorithm with access cost no larger than $2r \, \mathsf{E}[\mathsf{AccessCost}(\mathsf{MWU})]$. However, such an algorithm may incur an unbounded moving cost; even small changes in the distribution of MWU could lead to very different permutations after rounding. To deal with that, we update the permutation of Lazy-Rounding only if there are substantial changes in the distribution of MWU. Intuitively, small changes in

Algorithm 3 Lazy Rounding

Input: Sequence of requests $(S_1, \ldots, S_m)$ and the initial permutation $p_0 \in [n!]$.
Output: A permutation p_t at each round t, which serves request S_t.

start-phase $\leftarrow 1$
$P^1 \leftarrow$ uniform distribution over permutations
for each round $t \geq 1$ **do**
  if $d_{tv}(P^t, P^{\text{start-phase}}) \geq 1/n$ **then**
    $p_t \leftarrow p_{t-1}$
  else
    $p_t \leftarrow$ Greedy-Rounding(P^t)
    start-phase $\leftarrow t$
  end if
  Serve request S_t using permutation p_t.
  $w_p^{t+1} = w_p^t \cdot e^{-p(S_t)/n^3}$, for all permutations $p \in [n!]$.
  $P^{t+1} \leftarrow$ Distribution on permutations of MWU, $P_p^{t+1} = w_p^t / (\sum_{p \in [n!]} w_p^t)$.
end for

MWU's distribution should not affect much the access cost (this is formalized in Lemma 3.5). Moreover, Lazy-Rounding switches to a different permutation only if it is really required, which we use to bounds Lazy-Rounding's moving cost.

Bounding the Access Cost. We first show that the access cost of the algorithm Lazy-Rounding is within a factor of $4r$ from the expected access cost of MWU (Lemma 3.6). To this end, we first show that if the total variation distance between two distributions is small, then sampling from those distributions yields roughly the same expected access cost for any request S. The proof of the following is based on the optimal coupling lemma and can be found in the full version of this paper.

Lemma 3.5. *Let d and d' be two probability distributions over permutations. If that $d_{TV}(d, d') \leq 1/n$, for any request set S of size r, we have that*

$$\mathop{E}_{p \sim d'}[p(S)] \leq 2 \mathop{E}_{p \sim d}[p(S)].$$

Proof. By coupling lemma there exists a coupling (X, Y) such that

$$\Pr[X \neq Y] = d_{TV}(d, d'). \tag{3.2}$$

Clearly, $\mathsf{E}[\text{AccessCost}(X, S)] = \mathsf{E}_{p \sim d}[p(S)]$ and $\mathsf{E}[\text{AccessCost}(Y, S)] = \mathsf{E}_{p \sim d'}[p(S)]$. Note that since minimum cost for serving a request is 1 and maximum n, it will always hold that

$$1 \leq \mathsf{E}[\text{AccessCost}(X, S)], \mathsf{E}[\text{AccessCost}(Y, S)] \leq n.$$

We will show that $\mathsf{E}[\mathsf{AccessCost}(Y,S) \quad \mathsf{AccessCost}(X,S)] \quad 1$. This implies the lemma as follows:

$$\begin{aligned} {}_{p}\mathsf{E}_{d'}[p(S)] &= \mathsf{E}[\mathsf{AccessCost}(Y,S)] \\ &\quad \mathsf{E}[\mathsf{AccessCost}(X,S)] + 1 \quad 2\,\mathsf{E}[\mathsf{AccessCost}(X,S)] \\ &= 2\,{}_{p}\mathsf{E}_{d}[p(S)]. \end{aligned}$$

Thus it remains to show that $\mathsf{E}[\mathsf{AccessCost}(Y,S) \quad \mathsf{AccessCost}(X,S)] \quad 1$. For notational convenience, let the random variable $Z = \mathsf{AccessCost}(X,S) \quad \mathsf{AccessCost}(Y,S)$. It suffices to show that $\mathsf{E}[Z] \quad 1$. We have that

$$\begin{aligned} \mathsf{E}[Z] \quad & \Pr[\mathsf{AccessCost}(X,S) = \mathsf{AccessCost}(Y,S)] \quad 0 \\ & + \Pr[\mathsf{AccessCost}(X,S) \neq \mathsf{AccessCost}(Y,S)] \quad n, \end{aligned} \tag{3.3}$$

since whenever $X \neq Y$, the difference in the cost is upper bounded by n.

Observe that $\Pr[X(S) \neq Y(S)] \quad \Pr[X \neq Y]$; this is because if $X = Y$, then $\mathsf{AccessCost}(X,S) = \mathsf{AccessCost}(Y,S)$, but it may happen that $X \neq Y$ but $\mathsf{AccessCost}(X,S) = \mathsf{AccessCost}(Y,S)$. From (3.3) we get that

$$\mathsf{E}[Z] \quad \Pr[X \neq Y] \quad n. \tag{3.4}$$

Combining (3.2) with (3.4) we get that

$$\mathsf{E}[Z] \quad d_{\mathrm{TV}}(d,d') \quad n \quad \frac{1}{n} \quad n = 1. \qquad \square$$

We are now ready to upper bound the access cost of our algorithm.

Lemma 3.6. $\mathsf{AccessCost}(\textsc{Lazy-Rounding}) \quad 4r \quad \mathsf{E}[\mathsf{AccessCost}(\mathsf{MWU})]$.

Proof. Consider a phase of Lazy-Rounding starting at time t_1. We have that at any round $t \quad t_1$, $p_t = \text{Greedy-Rounding}(\mathsf{P}^{t_1})$, as long as $\mathrm{d_{TV}}(\mathsf{P}^t, \mathsf{P}^{t_1}) \quad 1/n$. By Theorem 3.6 and Lemma 3.5, we have that,

$$\mathsf{AccessCost}(\textsc{Lazy-Rounding}(t)) = p_t(S_t) \quad 2r \quad {}_{p}\mathsf{E}_{\mathsf{P}^{t_1}}[p(S_t)] \quad 4r \quad {}_{p}\mathsf{E}_{\mathsf{P}^{t}}[p(S_t)].$$

Overall we get,

$$\mathsf{AccessCost}(\textsc{Lazy-Rounding}) = \sum_{t=1}^{m} p_t(S_t) \quad 4r\,\mathsf{E}[\mathsf{AccessCost}(\mathsf{MWU})]. \qquad \square$$

Bounding the Moving Cost. We now show that the moving cost of Lazy-Rounding is upper bounded by the expected access cost of MWU.

Lemma 3.7. MovingCost(Lazy-Rounding) E[AccessCost(MWU)].

Proof. Lazy-Rounding moves at the end of a phase incurring a cost of at most n^2. Let t_1 and t_2 be the starting times of two consecutive phases. By the definition of Lazy-Rounding, $d_{\mathrm{TV}}(P^{t_1}, P^{t_2}) > 1/n$. By Lemma 3.3, we have that the access cost of MWU during t_1 and t_2 is at least n^2. We get that

$$\frac{\mathrm{MovingCost(ALG)}}{\mathrm{E[AccessCost(MWU)]}} \quad \frac{n^2\#\text{ different phases}}{n^2\#\text{different phases}} = 1. \quad \square$$

Theorem 3.2 follows from lemmas 3.6, 3.7 and 3.1. The details can be found in the full version.

Proof of Theorem 3.2. We now give the formal proof of the competitive ratio of Lazy-Rounding algorithm.

Theorem 3.2. *There exists a* $(5r + 2)$*-competitive deterministic online algorithm for the static version of the Online Min-Sum Set Cover problem.*

Proof. We show that our algorithm ALG is $(5r + 2)$-competitive.

By lemmata 3.6 and 3.7 we get that

$$\mathrm{Cost(Lazy\text{-}Rounding)} \quad (4r + 1) \quad \mathrm{AccessCost(MWU)}. \tag{3.5}$$

Now, we connect the access cost of MWU to the optimal cost. By Lemma 3.1 we have that

$$\mathrm{AccessCost(MWU)} \quad \frac{5}{4} \quad \mathrm{Cost(OPT)} + 2n^4 \ln n. \tag{3.6}$$

By (3.5) and (3.6) we get that

$$\mathrm{Cost(Lazy\text{-}Rounding)} \quad (5r + 2) \quad \mathrm{Cost(OPT)} + 2 \quad (4r + 1) \quad n^4 \ln n \qquad \square$$

On our Approach. Note that to a large extent, our approach is generic and can be used to provide static optimality for a wide range of online problems. Consider any MTS problem with N states such that in each request the service cost of each state is at most $C_{\max}$ and the diameter of the state space is D. Assume that there exists a rounding scheme providing a derandomization of MWU such that the service cost is within a factor of the expected

Algorithm 4 Move-All-Equally

Input: A request sequence $(S_1, \ldots, S_m)$ and the initial permutation $p_0 \; 2 \; [n!]$
Output: A permutation p_t at each round t.

1: **for** each round $t \quad 1$ **do**
2: $k_t \quad \min\{i \mid p_{t-1}[i] \; 2 \; S_t\}$
3: Decrease the index of all elements of S_t by $k_t \quad 1$.
4: **end for**

service cost of MWU. We explain how our technique from Section 3.3 can be used to obtain a $O(a)$-competitive algorithm against the best state.

Algorithm. The algorithm is essentially the same as Algorithm 3, using the MWU algorithm with learning rate $1/(D \quad C_{\max})$ and moving when the total variation distance between the distributions exceeds $1/C_{\max}$.

This way, Lemma 3.2 would give a bound of

$$d_{\mathrm{TV}}(P^t, P^{t+1}) \quad \frac{1}{D} \quad \mathrm{AccessCost}(\mathrm{MWU}(t))$$

and as a consequence in Lemma 3.3 we will get that the (expected) cost of MWU during a phase is at least D. Since the algorithm moves only at the end of the phase, incurring a cost of at most D, Lemma 3.7 will still hold. Last, it is easy to see that Lemma 3.5 will then hold for $d_{\mathrm{TV}}(d, d^0) \quad 1/C_{\max}$. This way, Lemma 3.6 would give that

$$\mathrm{AccessCost}(\mathrm{ALG}) \quad 2a \, \mathrm{E}[\mathrm{Cost}(\mathrm{MWU})].$$

Combining the above, we get that the algorithm is $O(a)$-competitive.

3.4 A Memoryless Algorithm

In this section we focus on memoryless algorithms. We present an algorithm, called Move-All-Equally (MAE), which seems to be the "right" memoryless algorithm for online MSSC. MAE decreases the index of all elements of the request S_t at the same speed until one of them reaches the first position of the permutation (see Algorithm 4). Note that MAE belongs to the *Move-to-Front* family, i.e., it is a generalization of the classic MTF algorithm for the list update problem. MAE admits two key properties that substantially differentiate it from the other algorithms in the Move-to-Front family presented in Section 3.2.

1. Let e_t denote the element used by OPT to cover the request S_t. MAE always moves the element e_t towards the beginning of the permutation.

2. It balances moving and access costs: if the access cost at time t is k_t, then the moving cost of MAE is roughly $r \cdot k_t$ (see Algorithm 4). The basic idea is that the moving cost of MAE can be compensated by the decrease in the position of element e_t. This is why it is crucial all the elements to be moved with the same speed.

Lower Bound. First, we show that this algorithm, besides its nice properties, fails to achieve a tight bound for the online MSSC problem.

In the lower bound instance, the adversary always requests the last r elements of the algorithm's permutation. Since MAE moves all elements to the beginning of the permutation, we end up in a request sequence where n/r disjoint sets are repeatedly requested. Thus the optimal solution incurs a cost of $\Theta(n/r)$ per request, while MAE incurs a cost of $\Omega(n \cdot r)$ per request (the details are in the full version) . Note that in such a sequence, MAE loses a factor of r by moving all elements, instead of one. However, this extra movement seems to be the reason that MAE outperforms all other memoryless algorithms and avoids poor performance in trivial instances, like other MTF-like algorithms.

Theorem 3.3. *The competitive ratio of the Move-All-Equally algorithm is* $\Omega(r^2)$.

Proof. MAE is given an initial permutation p_0 with size n, where $n = r \cdot k$, for integers r, k both greater than 1. At each round t, the adversary gives requests S_t, which consist of the last r elements in the permutation p_{t-1} of MAE. Since MAE moves all the elements of S_t to the beginning of p_t, the request S_{t+1} contains the r elements preceding the elements of S_t in p_t. Thus, the request sequence can be divided in m/k requests containing the same k pairwise disjoint requests, denoted as $S = \{S_1, \ldots S_k\}$. The optimal static solution can serve the request sequence by using only k elements, where each of these elements belongs to exactly one of the $S_j, 1 \leq j \leq k$. OPT has these elements in the first k positions and pays $1 + 2 + \ldots + k \leq k^2$ for every k consecutive requests $(S_1, \ldots S_k)$. MAE clearly pays $n - r + 1$ access cost and $r \cdot (n - r + 1)$ moving cost on every request, therefore $\text{MAE} = m \cdot (r+1) \cdot (n - r + 1)$. Then, the competitive ratio of MAE is at least

$$\frac{\text{Cost(MAE)}}{\text{Cost(OPT)}} \geq \frac{m \cdot (r+1) \cdot (n - r + 1)}{(m/k) \cdot k^2} = \frac{(r+1) \cdot (r \cdot k - r + 1)}{k} = \Omega(r^2) \quad \square$$

Upper Bounds. Let L denote the set of elements used by the optimal permutation on a request sequence such that $|L| = \ell$. That means, OPT has those ℓ elements in the beginning of its permutation, and it never uses the remaining $n - \ell$ elements. Consider a potential function $\Phi(t)$ being the number of inversions between elements of L and $U \setminus L$ in the permutation of MAE (an inversion occurs when an element of L is behind an element of $U \setminus L$). Consider the request S_t at time t and let k_t be the access cost of MAE.

Let e_t be the element used by OPT to serve S_t. Clearly, in the permutation of MAE, e_t passes (i.e., changes relative order w.r.t) k_t 1 elements. Among them, let L be the set of elements of L and R the elements of $U \setminus L$. Clearly, $|L| + |R| = k_t$ 1 and $|L|$ $|L| = \ell$. We get that the move of e_t changes the potential by $|R|$. The moves of all other elements increase the potential by at most $(r$ 1) ℓ. We get that

$$k_t + F(t) \quad F(t \quad 1) \qquad |L| + |R| \quad |R| + (r \quad 1)\ell \quad |L| + (r \quad 1)\ell \quad r\ell.$$

Since the cost of MAE at step t is no more than $(r + 1)$ k_t, we get that the amortized cost of MAE per request is $O(r^2 \ \ell)$. This implies that for all sequences such that OPT uses all elements of L with same frequencies (i.e, the OPT pays on average $\Omega(\ell)$ per request), MAE incurs a cost within $O(r^2)$ factor from the optimal. Recall that all other MTF-like algorithms are $\Omega(\sqrt{n})$ competitive even in instances where OPT uses only one element!

While this simple potential gives evidence that MAE is $O(r^2)$-competitive, it is not enough to provide satisfactory competitiveness guarantees. We generalize this approach and define the potential function $F(t) = \sum_{j=1}^{n} a_j \ p_t(j)$, where $p_t(j)$ is the position of element j at round t and a_j are some non-negative coefficients. The potential we described before is the special case where $a_j = 1$ for all elements of L and $a_j = 0$ for elements of $U \setminus L$.

By refining further this approach and choosing coefficients a_j according to the frequency that OPT uses element j to serve requests (elements of high frequency are "more important" so they should have higher values a_j), we get an improved upper bound.

Theorem 3.7. *The competitive ratio of* MAE *algorithm is at most* $2^{O(\sqrt{\log n \log r})}$.

Note that this guarantee is $o(n^e)$ and $\omega(\log n)$. The proof is based on the ideas sketched above but the analysis is quite involved. The main technical contribution of this section is proving the following lemma, from which Theorem 3.7 follows.

Lemma 3.8. *For any parameter* $b > 1$,

$$\text{Cost(MAE)} \quad 4b(2r)^{k+1} \ \text{Cost(OPT)} + (2r)^k n^2$$

where $k = \lceil 2 \log n / \log b \rceil$.

Note that Theorem 3.7 follows from Lemma 3.8 by balancing the values of b and $(2r)^{k+1}$. It is easy to verify that by setting $b = 2^{\sqrt{8 \log n \log 2r}}$, we obtain a competitive ratio at most $b^2 \quad 2^{8\sqrt{\log n \log r}}$.

Notation and Definitions. Let the frequency of an element e be the fraction of requests served by e in the optimal permutation. At a high-level, we divide the optimal permutation into $k + 1$ *blocks* where in each of the first k blocks the frequencies of all elements are

within a factor of b and the last block contains all elements with frequencies at most $1/n^2$. By construction, in worst-case $k \leq \frac{\log n}{\log b}$. More formally:

Let $f_j \in [0,1]$ denote the *covering frequency*, which is the total fraction of requests served by the optimal permutation with element j. For convenience, we reorder the elements such that $f_1 \geq f_2 \geq \ldots \geq f_n$. As a result, $\mathsf{Cost(OPT)} = m \cdot \sum_{j=1}^{n} j \cdot f_j$.

We partition the elements of U into $k+1$ *blocks*, as follows. The last block, E_{k+1} contains all elements j with $f_j \leq 1/n^2$; intuitively those are the least important elements. The block E_i for $1 \leq i \leq k$ contains all elements with frequencies in the interval $[f_1/b^{i-1}, f_1/b^i)$. Note that in worst case, $k = \lceil 2\log n / \log b \rceil$; we need that $f_1/b^k \leq 1/n^2$, which is equivalent to $k \geq f_1 \cdot \frac{2\log n}{\log b}$.

Lower Bound on optimal cost. Using this structure, we can express a neat lower bound on the cost of the optimal static permutation, which is formally stated in the following lemma. Lemma 3.9 provides a lower bound on $\mathsf{Cost(OPT)}$ using the first k blocks. For the rest of this section, let $F_i = \bigcup_{j=1}^{i} E_j$ the set of all elements in the first i blocks, for $1 \leq i \leq k$. Let also $f^i_{\max}, f^i_{\min}$ denote the maximum and minimum covering frequencies in the set E_i.

Lemma 3.9. *For any request sequence the optimal cost is at least*

$$\mathsf{Cost(OPT)} \geq \frac{m}{2} \sum_{i=1}^{k} |E_i| \cdot |F_i| \cdot f^i_{\min}.$$

Proof. Using the definitions above, OPT uses element j exactly $m \cdot f_j$ times for a total cost $j \cdot m \cdot f_j$. To account for all elements of the same bucket E_i together, we underestimate this cost and for an element $j \in E_i$ we charge OPT only for $m \cdot f^i_{\min}$ $\leq m \cdot f_j$ requests. We get that:

$$\begin{aligned}
\mathsf{Cost(OPT)} &= m \sum_{j=1}^{n} f_j \cdot j \geq m \sum_{i=1}^{k} \sum_{j \in E_i} f_j \cdot j \geq m \sum_{i=1}^{k} f^i_{\min} \sum_{j \in E_i} j \\
&= m \sum_{i=1}^{k} f^i_{\min} \sum_{j=1}^{|E_i|} (j + |F_{i-1}|) \\
&= m \sum_{i=1}^{k} f^i_{\min} \left(|E_i||F_{i-1}| + \frac{|E_i|(|E_i|+1)}{2} \right) \\
&\geq \frac{m}{2} \sum_{i=1}^{k} f^i_{\min} \cdot |E_i| \cdot (|F_{i-1}| + |E_i|) \\
&= \frac{m}{2} \sum_{i=1}^{k} f^i_{\min} \cdot |E_i| \cdot |F_i| \qquad \square
\end{aligned}$$

Upper Bound on cost of MAE. Using the blocks, we also obtain an upper bound on the cost of Move-All-Equally. This is formally stated in Lemma 3.10.

Lemma 3.10. *For any request sequence the cost of MAE algorithm can be upper bounded as follows:*

$$\text{Cost(MAE)} \quad (2r)^{k+1} \; m \sum_{i=1}^{k} |F_i||E_i| \; f^i_{\max} + 2(2r)^{k+1} \; m + (2r)^k \; n^2.$$

Before proceeding to the proof of Lemma 3.10, which is quite technically involved, we show how Lemma 3.8 follows from Lemmata 3.10 and 3.9.

Proof of Lemma 3.8. From Lemma 3.10 we have that

$$\begin{aligned} \text{Cost(MAE)} \quad & (2r)^{k+1} \; \underbrace{m \sum_{i=1}^{k} |F_i||E_i| \; f^i_{\max}}_{b \cdot \text{Cost(OPT)}} + \underbrace{2(2r)^{k+1} \; m}_{2(2r)^{k+1} \cdot \text{Cost(OPT)}} + (2r)^k \; n^2 \\ & 4b(2r)^{k+1} \; \text{Cost(OPT)} + (2r)^k \; n^2, \end{aligned}$$

where in the inequality we used the lower bound on Cost(OPT) from Lemma 3.9 and that $f^i_{\max} = f^i_{\min} \; b$ for all i. □

Proof of Lemma 3.10

It remains to prove Lemma 3.10 which gives the upper bound on the cost of MAE algorithm. The proof lies on the right selections of the coefficients a_j in the potential function $\Phi(t)$:

$$\Phi(t) = \sum_{j=1}^{n} a_j \; p_t(j),$$

where $p_t(j)$ is the position of element j at round t and a_j are some non-negative coefficients. More precisely, if $j \in E_i$ then $a_j = (2r)^{k-i}$ for $i = 1, \ldots, k$ and $a_j = 0$ if $j \in E_{k+1}$.

At time t, let k_t denote the access cost of MAE and let e_t be the element used by OPT to serve request S_t. Using the coefficients mentioned above, we can break the analysis into two different types of requests: (i) requests served by OPT using an element $e_t \in E_i$ for $1 \le i \le k$ and (ii) requests served by OPT using $e_t \in E_{k+1}$. At a high-level the first case is the one where the choice of coefficients is crucial; for the second, we show that the frequencies are so "small", such that even if MAE incurs an access cost of n, this does not affect the bound of Theorem 3.7.

We start with the first type of requests. We show the following lemma.

Lemma 3.11. *At time t, if $e_t \in E_i$, for $1 \le i \le k$, then we have that*

$$\mathsf{AccessCost}(\mathsf{MAE}(t)) + \mathsf{F}(t) - \mathsf{F}(t-1) \le (2r)^k \cdot |F_i|$$

For the second type of requests we have the following lemma.

Lemma 3.12. *The total amortized cost of* MAE *algorithm for all requests such that $e_t \in E_{k+1}$ is at most $(2r)^k m + m$.*

The proofs of those Lemmas are at the end of this section. We now continue the proof of Lemma 3.10.

Proof of Lemma 3.10. We upper bound the total access cost of MAE, that is $\sum_{t=1}^{m} k_t$. Clearly, by construction of MAE, the total cost is at most $(r+1) \cdot \sum_{t=1}^{m} k_t$. We have that:

$$\begin{aligned}
&\sum_{t=1}^{m} k_t + \mathsf{F}(t) - \mathsf{F}(t-1) \\
&= \sum_{i=1}^{k} \left(\sum_{t:e_t \in E_i} k_t + \mathsf{F}(t) - \mathsf{F}(t-1) + \sum_{t:e_t \in E_{k+1}} k_t + \mathsf{F}(t) - \mathsf{F}(t-1) \right) \\
&\le \sum_{i=1}^{k} \sum_{t:e_t \in E_i} (2r)^k \cdot |F_i| + \left((2r)^k + 1\right) \cdot m \\
&= \sum_{i=1}^{k} (2r)^k \cdot |F_i| \cdot m \cdot \sum_{j \in E_i} f_j^t + \left((2r)^k + 1\right) \cdot m \\
&\le \sum_{i=1}^{k} (2r)^k \cdot |F_i| \cdot m \cdot \sum_{j \in E_i} f_{\max}^i + \left((2r)^k + 1\right) \cdot m \\
&= \sum_{i=1}^{k} (2r)^k \cdot |F_i| \cdot |E_i| \cdot f_{\max}^i \cdot m + \left((2r)^k + 1\right) \cdot m,
\end{aligned}$$

where the first inequality comes from Lemmata 3.11, 3.12. Using $\mathsf{F}(0) \le (2r)^k \cdot 1 \cdot n^2$ we get

$$\begin{aligned}
\mathsf{Cost}(\mathsf{MAE}) &\le 2r \cdot \sum_{t=1}^{m} k_t \\
&\le 2r \left(\sum_{i=1}^{k} (2r)^k \cdot |F_i| \cdot |E_i| \cdot f_{\max}^i \cdot m + \left((2r)^k + 1\right) \cdot m + (2r)^k \cdot 1 \cdot n^2 \right) \\
&\le (2r)^{k+1} \sum_{i=1}^{k} |F_i| \cdot |E_i| \cdot f_{\max}^i \cdot m + 2(2r)^{k+1} \cdot m + (2r)^k \cdot n^2
\end{aligned}$$

□

We conclude the Section with the proofs of lemmata 3.11, 3.12, which were omitted earlier.

Proof of Lemma 3.11. At time t, the access cost of MAE is k_t. We have that

$$\begin{aligned} k_t + \mathsf{F}(t) \quad \mathsf{F}(t \quad 1) &= k_t \quad a_{e_t} \quad k_t + \sum_{j \in e_t} a_j(p_t(j) \quad p_{t \ 1}(j)) \\ &\quad k_t \ (1 \quad a_{e_t}) + (2r)^{k \ 1} \ |F_i| \ r \\ &\quad + \sum_{j \in U \setminus F_i} a_j(p_t(j) \quad p_{t \ 1}(j)), \end{aligned}$$

where the inequality follows from the fact that each element in F_i can increase its position by at most r and that the maximum coefficient a_j is at most $(2r)^{k \ 1}$. We complete the proof by showing that

$$k_t \ (1 \quad a_{e_t}) + \sum_{j \in U \setminus F_i} a_j(p_t(j) \quad p_{t \ 1}(j)) \quad 0$$

If $e_t \in E_k$ then $a_{e_t} = 1$ and $a_j = 0$ for all $j \in U \setminus F_k$, thus the left hand side equals 0 and the inequality holds. It remains to analyze the case where $e_t \in E_i$ and $i < k$.

$$\begin{aligned} & k_t \ (1 \quad a_{e_t}) + \sum_{j \in U \setminus F_i} a_j(p_t(j) \quad p_{t \ 1}(j)) \\ & \quad k_t \ (1 \quad a_{e_t}) + \sum_{j \in U \setminus F_i} \frac{a_{e_t}}{2r} \ |p_t(j) \quad p_{t \ 1}(j)| \\ & \quad k_t \ (1 \quad a_{e_t}) + \frac{a_{e_t}}{2r} \ r \ k_t \\ & = k_t \ (1 \quad a_{e_t} + \frac{a_{e_t}}{2}) \quad 0. \end{aligned}$$

The last inequality is due to the fact that $a_{e_t} \quad r$, thus $1 \quad a_{e_t}/2$ is negative if $r \quad 2$. □

Proof of Lemma 3.12. We sum the total amortized cost over all time steps t such that $e_t \in E_{k+1}$. We have that

$$\begin{aligned} & \sum_{t : e_t \in E_{k+1}} k_t + \mathsf{F}(t) \quad \mathsf{F}(t \quad 1) \\ & \quad \sum_{t : e_t \in E_{k+1}} n + (2r)^{k \ 1} \ n \ r \quad \left((2r)^k + 1\right) \ n \sum_{t : e_t \in E_{k+1}} 1 \\ & \quad \left((2r)^k + 1\right) \ n \ m \sum_{j \in E_{k+1}} f_j \quad \left((2r)^k + 1\right) \ n \ m \ \frac{|E_{k+1}|}{n^2} \\ & \quad \left((2r)^k + 1\right) \ m \end{aligned}$$

□

3.5 Dynamic Online Min-Sum Set Cover

In this section, we turn our attention to the dynamic version of online MSSC. In online dynamic MSSC, the optimal solution maintains a trajectory of permutations $p_0, p_1, \ldots, p_t, \ldots$

and use permutation p_t to serve each request S_t. The cost of the optimal dynamic solution is $\mathrm{OPT}_{\text{dynamic}} = \sum_t (p_t(S_t) + d_{\mathrm{KT}}(p_{t-1}, p_t))$, where $\{p_t\}_t$ denotes the optimal permutation trajectory for the request sequence that minimizes the total access and moving cost.

We remark that the ratio between the optimal static solution and the optimal dynamic solution can be as high as $\Omega(n)$. For example, in the sequence of requests $\{1\}^b \{2\}^b \ldots \{n\}^b$, the optimal static solution pays $\Omega(n^2 b)$, whereas the optimal dynamic solution pays $\Omega(n^2 + n - b)$ by moving the element that covers the next $n - b$ requests to the first position and then incurring access cost 1. The above example also reveals that although Algorithm 3 is $\Omega(r)$-competitive against the optimal static solution, its worst-case ratio against a dynamic solution can be $\Omega(n)$.

Next, we investigate the competitive ratio of *Move-All-Equally* (MAE) algorithm from Section 3.4 against the dynamic adversary. We begin with an upper bound:

Theorem 3.4. *The competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is* $O(r^{3/2}\sqrt{n})$.

The approach for proving the upper bound is generalizing that exhibited in Section 3.4 for the static case. We use a generalized potential function $\Phi(t) = \sum_{j=1}^{n} a_j^t \cdot p_t(j)$; i.e, the multipliers a_j may change over time so as to capture the moves of $\mathrm{OPT}_{\text{dynamic}}$. To select coefficients a_j^t we apply a two-level approach. We observe that there is always a 2-approximate optimal solution that moves an element of S_t to the front (similar to classic MTF in list update). We call this $\mathrm{MTF}_{\mathrm{OPT}}$. We compare the permutation of the online algorithm with the permutation maintained by this algorithm; at each time, elements the beginning of the offline permutation are considered to be "important" and have higher coefficients a_j^t.

We will use the above arguments to bound the total cost of MAE. Recall that the access cost of MAE at round t is k_t. We have that at each round $t \geq 1$,

$$\begin{aligned} k_t + \Phi(t) - \Phi(t-1) &= k_t + \sum_{j=1}^{n} a_j^t \cdot p_t(j) - \sum_{j=1}^{n} a_j^{t-1} \cdot p_{t-1}(j) \\ &= (1 - a_{e_t}^t) \cdot k_t + \sum_{j \neq e_t} a_j^t \cdot (p_t(j) - p_{t-1}(j)) \\ &\quad + \sum_{j=1}^{n} (a_j^t - a_j^{t-1}) \cdot p_{t-1}(j) \\ &\leq r - c + \underbrace{\left(\sum_{j=1}^{n} a_j^t - a_j^{t-1} \right)}_{\text{bounded by } \mathrm{MTF}_{\mathrm{OPT}}(t)} \cdot n, \end{aligned} \tag{3.7}$$

where in the last inequality we used that $a_{e_t}^t = 1$ and that for each element, its position change from round $t-1$ to round t $p_t(j) - p_{t-1}(j)$ can be at most r.

The following lemma shows an upper bound on the quantity $\sum_{j=1}^{n} a_j^t - a_j^{t-1}$.

Lemma 3.13. *The term* $\sum_{j=1}^{n}(a_j^t - a_j^{t-1})$*, which appears in the difference* $\Phi(t) - \Phi(t-1)$ *of the potential function* $\Phi(t)$ *can be bounded by the moving cost of the* $\mathrm{MTF}_{\mathrm{OPT}}(t)$ *as follows:* $\sum_{j=1}^{n}(a_j^t - a_j^{t-1}) - a_{e_t}^t - a_{e_t}^{t-1} \leq \mathrm{MovingCost}(\mathrm{MTF}_{\mathrm{OPT}}(t))/c$.

Proof. For the first part of the inequality, observe that from round $t-1$ to round t, $\mathrm{MTF}_{\mathrm{OPT}}$ only moves element e_t towards the beginning of the permutation. The latter implies, that for all elements $j \neq e_t$, $a_j^t - a_j^{t-1}$. For the second part of the inequality, observe that if $a_{e_t}^{t-1} = 0$, then by the definition of the coefficients, e_t does not belong in the first c positions (of $\mathrm{MTF}_{\mathrm{OPT}}$'s permutation) at round $t-1$. Since at round t, $\mathrm{MTF}_{\mathrm{OPT}}$ moves e_t in the first position of the list, $\mathrm{MovingCost}(\mathrm{MTF}_{\mathrm{OPT}}(t)) - c$ □

Notice that the overall access cost of both MAE and $\mathrm{MTF}_{\mathrm{OPT}}$ is just m since at each round t both of the algorithms admit an element, covering S_t, in the first position of the permutation. As a result by setting $c = \sqrt{n/r}$ and applying Lemma 3.13,

$$\begin{aligned}
\mathrm{Cost}(\mathrm{MAE}) &= \mathrm{AccessCost}(\mathrm{MAE}) + \mathrm{MovingCost}(\mathrm{MAE}) \\
&= m + r\sum_{t=1}^{m} k_t \\
&\quad m + 2r^{3/2}\sqrt{n}\sum_{t=1}^{m}\mathrm{MovingCost}(\mathrm{MTF}_{\mathrm{OPT}}(t)) + \Phi(0) - \Phi(m) \\
&\quad 2r^{3/2}\sqrt{n}\ \mathrm{Cost}(\mathrm{MTF}_{\mathrm{OPT}}) + \Phi(0) - \Phi(m) \\
&\quad 2r^{3/2}\sqrt{n}\,\mathrm{Cost}(\mathrm{MTF}_{\mathrm{OPT}}).
\end{aligned}$$

The first inequality follows from inequality (3.7). The last inequality follows from the fact that $\Phi(0) - \Phi(m)$ since at round 0 the permutations of MAE and $\mathrm{MTF}_{\mathrm{OPT}}$ are the same. We complete the section with the proof of Lemma 3.14, stating that $\mathrm{Cost}(\mathrm{MTF}_{\mathrm{OPT}})$ $2\ \mathrm{Cost}(\mathrm{OPT}_{\mathrm{dynamic}})$.

Lemma 3.14. *For any sequence of requests* $(S_1, \ldots, S_m)$*,*

$$\mathrm{Cost}(\mathrm{MTF}_{\mathrm{OPT}}) \quad 2\ \mathrm{Cost}(\mathrm{OPT}_{dynamic})$$

Proof. To simplify notation, let x_t and y_t respectively denote the permutations of $\mathrm{MTF}_{\mathrm{OPT}}$ and $\mathrm{OPT}_{\mathrm{dynamic}}$ at round t. We will use as potential the function $\Phi(t) = \mathrm{d_{KT}}(x_t, y_t)$ i.e. the number of inverted pairs between the permutation x_t and y_t. Let L_t denote the set of elements that are on the left of e_t in permutation x_t and on the left of e_t in permutation y_t. Respectively, R_t denotes the set of elements that are on the left of e_t in permutation x_t, but on the right of e_t in permutation y_t. Clearly, at round t, $\mathrm{MTF}_{\mathrm{OPT}}$ pays $L_t + R_t$ for moving

cost and 1 for accessing cost. At same time, $\mathsf{OPT}_{\text{dynamic}}$ pays at least $L_t + 1$ for accessing cost and $d_{KT}(y_t, y_{t-1})$ for moving cost.

$$\begin{aligned}\mathsf{Cost}(\mathsf{MTF}_{\mathsf{OPT}}) + \Phi(t) - \Phi(t-1) &= L_t + R_t + 1 + d_{KT}(x_t, y_t) - d_{KT}(x_{t-1}, y_t) \\ &+ d_{KT}(x_{t-1}, y_t) - d_{KT}(x_{t-1}, y_{t-1}) \\ &\leq L_t + R_t + 1 + (L_t - R_t) \\ &+ d_{KT}(x_{t-1}, y_t) - d_{KT}(x_{t-1}, y_{t-1}) \\ &\leq 2(L_t + 1) + d_{KT}(y_t, y_{t-1})\end{aligned}$$

The first inequality follows by the definition of the of the sets R_t and L_t, while the second by the triangle inequality in the Kendall-tau distance $d_{KT}(\cdot, \cdot)$. The proof is completed by summing all over m and using the fact that $d_{KT}(x_0, y_0) = 0$. □

Next, we show an almost matching lower bound. The lower bound is based on a complicated adversarial request sequence; we sketch the main ideas. Let k be an integer. During a *phase* we ensure that:

1. There are $2k$ "important" elements used by OPT; we call them $e_1, \ldots, e_{2k}$. In the beginning of the phase, those elements are ordered in the start of the optimal permutation p^*, i.e., $p^*[e_j] = j$. The phase contains k consecutive requests to each of them, in order; thus the total number of requests is $2k^2$. OPT brings each element e_j at the front and uses it for k consecutive requests; thus the access cost of OPT is $2k^2$ (1 per request) and the total movement cost of OPT of order $\Theta(k^2)$. Over a phase of $2k^2$ requests, OPT incurs an overall cost $\Theta(k^2)$, i.e., an average of $O(1)$ per request.
2. The first $k + r - 2$ positions of the online permutation will be always occupied by the same set of "not important" elements; at each step the $r - 2$ last of them will be part of the request set and MAE will move them to the front. Thus the access cost will always be $k + 1$ and the total cost more than $(r + 1) \cdot k$.

The two properties above are enough to provide a lower bound $\Omega(r \cdot k)$; the optimal cost is $O(1)$ per request and the online cost $\Omega(r \cdot k)$. The goal of an adversary is to construct a request sequence with those two properties for the largest value of k possible.

The surprising part is that although MAE moves all requested elements towards the beginning of the permutation, it never manages to bring any of the "important" elements in a position smaller than $r + k - 2$. While the full instance is complex and described in the full version, at a high-level, we make sure that whenever a subsequence of k consecutive requests including element e_j begins, e_j is at the end of the online permutation, i.e., $p_t[e_j] = n$. Thus, even after k consecutive requests where MAE moves it forward by distance k, it moves by k^2 positions; by making sure that $n - k^2 > r + k - 2$ (which is true for

some $k = \mathsf{W}(\sqrt{p/n})$), we can make sure that e_j does not reach the first $r + k - 2$ positions of the online permutation.

Lower bound. Next, we prove the following theorem showing a lower bound on MAE, which nearly matches its upper bound.

Theorem 3.5. *For any $r \geq 3$, the competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is $\mathsf{W}(r\sqrt{p/n})$.*

The constructed request sequence $(S_1, \ldots, S_m)$ will have the following properties:

1. The size of every request S_t is $r - 3$.
2. Every request forces the algorithm to pay an access cost of $\mathsf{W}(\sqrt{p/n})$.
3. The request sequence contains $\mathsf{W}(\sqrt{p/n})$ consecutive requests that share a common element, which we call the *pivot*.

The first two properties ensure that MAE will pay $\mathsf{W}(r\sqrt{p/n})$ moving cost on every request. The third property will enable the optimal dynamic solution $\mathsf{OPT}_{\text{dynamic}}$ to pay 1 access cost for the consecutive requests that share the common element.

Before proceeding to formal details of the proof of Theorem 3.5, we will present the main ideas of the construction. To this end, let p_0 be the MAE's initial permutation and think of it as being divided into $k + 2$ blocks. The first block contains $k + r - 1$ elements and all other blocks contain k elements, therefore $n = k^2 + 2k + r - 1$. The "important" elements, which the optimal dynamic solution uses to *cover* all requests, are the elements of the second and last block initially. Meaning that the optimal dynamic solution can cover all requests using only the $2k$ "important" elements. Figure 3.1 depicts the structure of the initial permutation for $k = 3$ and $r = 3$, where "important" elements are colored red and green.

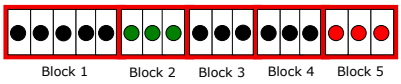


FIGURE 3.1: The structure in blocks for $k = 3$ and $r = 3$. Blocks are drawn with thick red borders. The elements that uses the optimal dynamic solution to cover the request sequence are colored green and red and lie in the second and last block initially .

The following definition formalizes the concept of blocks discussed above. Blocks are defined in terms of the indices of the permutation.

Definition 3.15. *Let $p = (p_0, \ldots, p_m)$ be a sequence of permutations, where each permutation has size $n = k^2 + 2k + r - 1$. We divide each permutation into $k + 2$ consecutive*

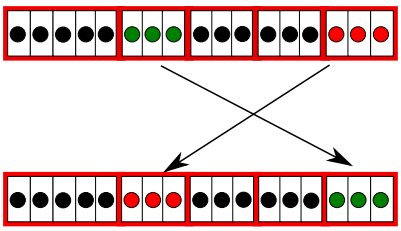


FIGURE 3.2: The permutation of MAE before the start of a round and after k rounds, for $k = 3$ and $r = 3$. The elements of the second block have moved to the last block and vice versa.

blocks $b_1, b_2, \ldots b_{k+2}$*, where a block is a set of consecutive indices in each permutation with* $|b_1| = k + r - 1$ *and* $|b_2| = |b_3| = \ldots = |b_{k+2}| = k$*. Therefore,* $b_1 = [1, \ldots, k + r - 1]$ *and* $b_i = [(i-1)k + r, \ldots, ik + r - 1]$*, for* $2 \le i \le k + 2$*.*

For ease of exposition, we will divide the analysis in phases, which are subdivided in rounds. The first round starts with the request S_1, which contains the last element of the permutation p_0 ($pivot_1$) of the MAE algorithm and ends when $pivot_1$ arrives in position $k + r$. Inductively, at the start of a round j, S_j contains the last element of permutation p_t and lasts until this element arrives in position $k + r$. A phase ends after $2k$ such rounds and a new phase begins. Formally:

Definition 3.16. *A phase defines the period, starting with* k *elements in block* b_2 *and* k *elements in block* b_{k+2} *and ending after* $2k$ *rounds of requests with these elements back in the blocks, where they started. A round* j *starts when the adversary requests the last element , denoted as* $pivot_j$*, of the current permutation of MAE (with index* $k^2 + 2k + r - 1$*) and ends when MAE places this element in the first position of the second block* b_2 *(with index* $k + r$*). The duration of the round is the number of requests from the start of the round until the end of the round and will be* $k + 1$*.*

In order to describe the request sequence, we give the color green to k elements and the color red to k other. These are the elements, used by $\mathsf{OPT}_{\text{dynamic}}$ and every request contains one of them. The rest of them are colored black. Let all the green elements be in the second block (b_2) and all red elements be in the last block (b_{k+2}) initially. The request sequence may seem complicated, however its constructed following two basic principles. The first is to decrease the position of the $pivot$ element (which is initially red) by k in the permutation and the second is to increase the position of the k green elements by one on every request. This is easily achieved by requests S_t of the form:

$$S_t = \{\underbrace{p_t(k+1), \ldots, p_t(k+r-2)}_{black}, \underbrace{\text{element succeeding the green block}}_{black}, \underbrace{\text{pivot}}_{red}\}$$

Then, at the end of round j, $pivot_j$ will be in the first position of the second block and the green elements will be in block $j + 2$. After k rounds, all red elements will be in the second block and all green elements will be in the last block, thus the adversary can repeat the request sequence starting again from round 1. Figure 3.2 depicts the positions of the "important" elements at the start of round 1 and at the end of round k.

The formal definition of the request sequence is shown in Algorithm 5.

Algorithm 5 Adversarial request sequence

Let $S_{ij} = [p_1, \ldots, p_r]$ be the i-th request of round j, where $p_1, \ldots, p_r$ denote the indices of the requested elements in the current permutation of MAE. The request sequence for k consecutive rounds is (every round has $k + 1$ requests):
Round 1. The ith request of round 1, for $1 \quad i < k$ is:
$S_{i1} = [k + 1, \ldots, k + r \quad 2, 2k + r \quad 1 + i, k^2 + 2k + r \quad 1 \quad (i \quad 1)k]$
and the two last requests of round 1 are:
$S_{k1} = [k + 1, \ldots, k + r \quad 1, 3k + r \quad 1]$ and $S_{(k+1)1} = [k, \ldots, k + r \quad 2, 2k + r \quad 1]$.
Round j from 2 to k. The ith request of round j, for $i \; 2 \; fk \quad j + 2, k + 1g$ is :
$S_{ij} = [k + 1, \ldots, k + r \quad 2, (j + 1)k + r \quad 1 + i, k^2 + 2k + r \quad 1 \quad (i \quad 1)k]$,
for the request i with $i = k \quad j + 2$ is :
$S_{ij} = [k + 1, \ldots, k + r \quad 1, (j + 1)k + r \quad 1]$
and for $i = k + 1$ is :
$S_{(k+1)j} = [k, \ldots, k + r \quad 2, 2k + r \quad 1]$.
Then, the request sequence starts again from round 1.

Notice that the last request of round j places $pivot_j$ in the first position of block b_2, just in front of pivot elements of previous rounds. This way it is guaranteed that all k red elements will be in b_2 after k rounds. Moreover, at each round $j > 1$ there is a request (the $(k \quad j + 1)$th) that increases the positions of $k \quad j + 2$ green elements by two, since they are passed from both the element succeeding them and the pivot element (the other $j \quad 2$ elements are passed only by the element succeeding the green block). Since, the adversary wants all green elements to be in block $j + 2$ after round j, the next request does not move the $k \quad j + 2$ green elements and the other $j \quad 2$ are moved because $pivot_j$ passes them (see Figure 3.3).

The following lemma shows that MAE will arrive in a symmetric permutation after k rounds.

Lemma 3.17. *Let p be a permutation of definition 3.15 at the start of a round and let $X = fx_1, x_2, \ldots, x_kg$ and $Y = fy_1, y_2, \ldots, y_kg$ be the elements in b_2 and b_{k+2} respectively before the start of a round. Then, after k rounds, MAE has moved the elements of Y in block b_2 and the elements of X in block b_{k+2}.*

Proof. We have to prove that after j rounds, j elements of Y will be in b_2 and all elements of X will be in b_{j+2}. Observe that the only way to increase the index of an element e in

a permutation by c is to move c elements with higher index than e in the permutation to positions with lower index than e (it increases by one if an element arrives at the position of e).

Let $pivot_j$ be the pivot element of round j, which by definition is the last element in the permutation at the start of a round. After k requests to $pivot_j$ involving also the $(k+1)$th element and elements in positions that do not increase the index of $pivot_j$, $pivot_j$ moves a total of k^2 positions to the left arriving in position $2k + r - 1$ of the current permutation. Then, the last request of this round will move it to the first position of b_2, moving the pivot element of the previous phase to the second position of b_2. By construction of the request sequence, $pivot_j$ is the jth element of Y requested so far. Therefore, at the end of round j, j elements of Y will be in the first j positions of b_2 .

We now show that all elements of X will be in b_{j+2} at end of round j. Particularly, we show inductively that at the beginning of jth round all elements of X are in block b_{j+1} and MAE moves all elements of X to block b_{j+2} when the jth round ends.

Induction Base: In the first round, k requests contain the element succeeding X, $r - 2$ elements in positions $k+1, \ldots, k+r-2$ of the permutation and an element, which has index higher than $k + r - 2$ (which is $pivot_1$) and does not change the positions of elements of X. The $(k+1)$th request does not change the positions of elements of X, since it contains elements with lower index. Therefore, the algorithm moves all elements of X exactly k positions to the right and they will be in b_3 when round 1 ends.

Inductive Step: For $j > 1$, we have $k - 1$ requests, where each of them forces the algorithm to move elements of X one position to the right. However, there is exactly one request S_{ij} with $i = k - j + 1$, where $k - j + 2$ elements of X increase their positions by two. These elements are passed by both the element succeeding X and the pivot element. The other $j - 2$ elements are passed only by the element succeeding X, thus increasing their positions by one. Therefore, the next request of the adversary (the $k - j + 2$-th) makes the algorithm move $pivot_j$ and the elements in positions $k+1, \ldots k+r-1$, therefore moving only the $j - 2$ elements of X one position to the right and the other elements of X remain in their positions.

We conclude that at each round the elements of X move to the next right block. Since, they are initially positioned in b_2, after k rounds they end up in block b_{k+2}. □

We are now ready to provide the proof of Theorem 3.5

Theorem 3.5. *For any $r - 3$, the competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is $\Omega(r\sqrt{n})$.*

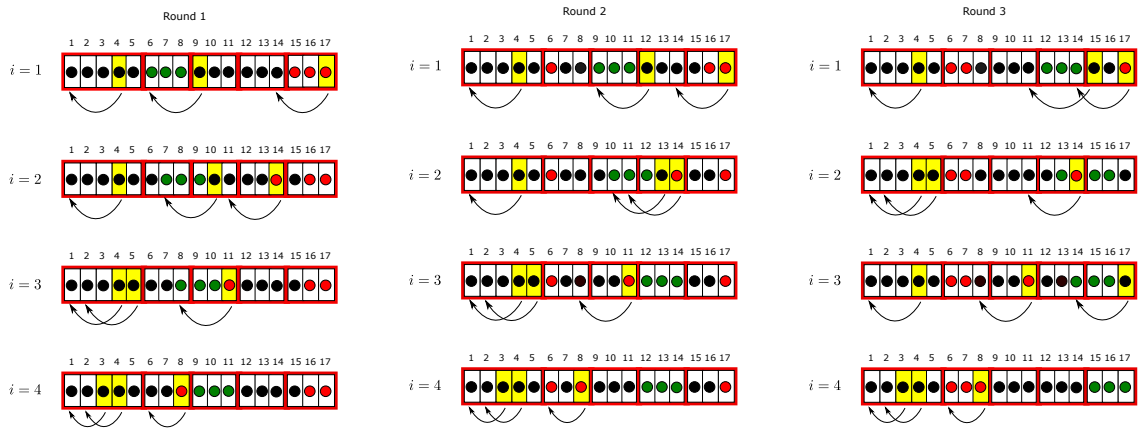


FIGURE 3.3: Execution of MAE on the adversarial sequence for $r = 3$ and $k = 3$. The requested positions in the list have yellow background and the arrows indicate the positions of the elements after every request. For round $j > 1$, the ith request with $i = k - j + 1$ makes MAE increase the position of $k - j + 2$ green elements by two. The next request is such that only the $j - 2$ remaining green elements increase their position by one.

Proof. We compute the costs paid by MAE and $\mathsf{OPT}_{\text{dynamic}}$ after k rounds of a phase, since the request sequence of Definition 5 is then repeated.

First, we bound the optimal cost. Initially, the optimal solution incurs a moving cost $O(k \cdot n)$ to move the $2k$ elements of b_2 and b_{k+2} in the first $2k$ positions of its permutation. Then, at the start of round j, it brings $pivot_j$ to the first position and incurs an access cost of 1 for $k + 1$ consecutive requests. So, after k rounds it pays at most $2k^2$ moving cost plus $k \cdot (k + 1)$ access cost, which sums to at most $4k^2$.

We now account the online cost. MAE pays $r \cdot k$ for the first k requests of each round and $r \cdot (k - 1)$ for the last request. So, the total cost for k rounds is $r \cdot k \cdot (k^2 + k - 1) > r \cdot k^3$. From Lemma 3.17, all elements of b_2 are in b_{k+2} and all elements of b_{k+2} are in b_2 after k rounds.

The adversary can repeat the same strategy to create an arbitrarily long request sequence. Let l be the number of times the same k-round strategy is applied. We get that

$$\frac{\text{Cost}(\mathsf{MAE})}{\text{Cost}(\mathsf{OPT}_{\text{dynamic}})} \quad \frac{l \cdot r \cdot k^3}{4l \cdot k^2 + O(k \cdot n)} \quad ! \quad \Omega(k).$$

The result follows for $l \quad ! \quad \infty$ and $k = \Omega(\sqrt{n})$.

□

4 The Online K-Facility Reallocation Problem on the Line

4.1 Introduction

In this chapter, we study the online variant of the K-Facility Reallocation problem on the real line and we focus mostly on the special case with $K = 2$ facilities.

Our first result is a lower bound of K for the online K-Facility Reallocation problem. We prove the following theorem, by constructing a reduction to the K-server problem.

Theorem 4.1. *Every c-competitive deterministic algorithm for the K-Facility Reallocation problem can be turned to a c-competitive deterministic algorithm for the K-server problem.*

Theorem 4.1 demonstrates that K-Facility Reallocation is harder than K-server. Since K-server has a lower bound of K in the deterministic competitive ratio, the same is also true for K-Facility Reallocation. From a technical viewpoint, the K-Facility Reallocation problem poses a new challenge, since it is *much* harder to track the movements of the optimal algorithm as the clients keep coming. It is not evident at all exactly how ideas from the K-server problem can be applied to the K-Facility reallocation problem, especially for more general metric spaces. As a first step towards this direction, we design a constant-competitive algorithm, when $K = 2$.

Theorem 4.2. *For $K = 2$, there exists a $O(1)$-competitive algorithm for the K-Facility Reallocation on the real line.*

Our algorithm appears in Section 4.3 and is inspired by the *double coverage algorithm* proposed for the K-server problem [54]. The online algorithm performs two steps at each stage. In Step 1, facilities are initially moved towards the positions of the clients. The purpose of this step is to bring at least one facility close to the clients. In Step 2, our algorithm determines the final positions of the two facilities. The algorithm decides if it will serve the clients with one facility or if it will use both facilities taking into account the tradeoff between the resulting connection and moving cost.

4.2 Lower Bound for Online K-Facility Reallocation

In this section, we prove that the online K-Facility Reallocation problem has a lower bound of K.

We start with the observation that the online K-Facility Reallocation problem with $K \quad 2$ facilities is a natural and interesting extension of the classical K-server problem, which has been a driving force in the development of online algorithms for decades. The key difference is that, in the K-server problem, there is a single client that changes her location at each stage and a single facility has to be relocated to this new location at each stage. Therefore, the total connection cost is by definition 0, and we seek to minimize the total moving cost.

The idea is to show that any algorithm for the K-server problem can be turned to an algorithm for the online K-Facility Reallocation problem with the same competitive ratio. Since the K-server problem has a lower bound of K, the same holds for the online K-Facility Reallocation problem

Theorem 4.1. *Every c-competitive deterministic algorithm for the K-Facility Reallocation problem can be turned to a c-competitive deterministic algorithm for the K-server problem.*

Proof. Consider an instance I of the K-server problem with requests $r_1, \dots r_T$ on the line. We construct an instance I^0 for the K-Facility Reallocation problem with 2 clients on the same metric with requests $r_1^0, \dots, r_T^0$, where each r_i^0 is an 2-dimensional ($n = 2$) of the form $r_t^0 = (r_t, r_t), 1 \quad t \quad T$. This essentially means that, at each round t, 2 clients are requested on the location on the line, where the request of the K-server problem for round t lies.

Now assume that we run a c-competitive algorithm ALG^0 for the K-Facility Reallocation problem problem on the instance I^0. ALG^0 can be transformed to an algorithm ALG for the K-server problem as follows: Let f_t denote the facility in I^0, which is closer to r_t^0 than any other facility and let s_t be the corresponding server in I. Then, this server moves on r_t to serve it and then returns to the position, where f_t lies (f_t may serve r_t^0 from a distance). All other servers are moved to the positions of their corresponding facilities.

Since the initial positions of facilities in I^0 and the servers in I are the same, all the servers will be on the same positions on the line as their corresponding facilities at the end of each round. Next, we analyze the costs paid by ALG and ALG^0 at each round t. All servers except s_t will move the same distance with their corresponding facilities, thus the cost paid for them is the same for ALG and ALG^0. Regarding f_t, assume that it moves a and connects r_t^0 from distance b. Then it pays $a + 2b$ overall to serve both clients. Additionally, the request r_t^0 is at distance $a + b$ from f_t at the start of round t. Then, s_t will pay $a + b$ for moving s_t on the corresponding request r_t and then b to move s_t to the position of f_t. Therefore, the cost paid for all servers of ALG at each round is the same with the cost paid for all facilities of ALG^0. Let OPT_I and OPT_{I^0} denote the optimal solutions of I and I^0 respectively. Since

any feasible solution for I is feasible for I', we have that $OPT_{I'} \geq OPT_I$ and therefore $Cost(ALG) = Cost(ALG') \leq Cost(OPT_{I'}) \leq Cost(OPT_I)$.

□

The next section is dedicated to prove Theorem 4.2. The constant in the competitive ratio is 63 and although it is possible to improve the competitive ratio of Algorithm 6 by a much more technically involved analysis, we stress here that it is not possible to turn the result into any constant factor, since the previous result rules out the existence of a deterministic algorithm for the 2-Facility Reallocation problem on the line with competitive ratio lower than 2.

4.3 A Constant Competitive Algorithm for the Online 2-Facility Reallocation Problem

In this section, we present a constant competitive algorithm for the online 2-Facility Reallocation problem and we discuss the core ideas that prove its performance guarantee.

The online algorithm, denoted as Algorithm 6, is inspired by the *double coverage algorithm* proposed for the K-server problem [54]. The *double coverage algorithm* solves the K-server problem on the line optimally in the sense that it achieves a competitive ratio of K, matching the lower bound for this problem. This simple algorithm performs one of the following steps based on the relevant positions of the facilities and the single client:

- If the client is located between two facilities, then it moves these facilities with the same speed towards the client until the closest one of them reaches the client.
- Else, the closest facility is moved on the client.

Notice that the *double coverage algorithm* moves at most 2 facilities towards the client. This helps us design the first step of our online algorithm for 2-Facility Reallocation problem, which performs the following two basic steps.

In Step 1, facilities are initially moved towards the positions of the clients. This step is also performed by the *double coverage algorithm* with one major difference. That is, we now have n clients, which define the interval $[a_1^t, a_n^t]$. Thus, this step ends, when we reach the leftmost (a_1^t) or the rightmost (a_n^t) client. We remark that in Step 1, the final positions of the facilities at stage t are not yet determined. The purpose of this step is to bring at least one facility close to the clients. Note that this step is not performed if a facility is already inside the interval $[a_1^t, a_n^t]$ at the beginning of stage t.

In Step 2, our algorithm determines the final positions of the facilities x_1^t, x_2^t. After Step 1, at least one of the facilities is inside the interval $[a_1^t, a_n^t]$, meaning that at least one of the

facilities is close to the clients. As a result, our algorithm may need to decide between moving the second facility close to the clients or just letting the clients connect to the facility that is already close to them. Obviously, the first choice may lead to small connection cost, but large moving cost, while the second has the exact opposite effect. Roughly speaking, Algorithm 6 does the following: If the connection cost of the clients, when placing just one facility optimally, is not much greater than the cost for moving the second facility inside $[a_1^t, a_n^t]$, then Algorithm 6 puts the first facility to the position that minimizes the connection cost, if one facility is used. Otherwise, it puts the facilities to the positions that minimize the connection cost, if two facilities are used. We formalize how this choice is performed, introducing some additional notation.

Definition 4.1.

- $C_t = \{a_1^t, \ldots, a_n^t\}$ *denotes the positions of the clients at stage* t *ordered from left to right.*
- *If* C *is a set of positions with* $|C| = 2k, k \in \mathbb{N}_{>0}$*, then* M_C *denotes the median interval of the set, which is the interval* $[a_{n/2}, a_{n/2+1}]$*. If* $|C| = 2k+1, k \in \mathbb{N}_0$*, then* M_C *is a single point.*
- $H(C)$ *denotes the optimal connection cost for the set* C *when all clients of* C *connect to just one facility. That is* $H(\emptyset) = 0$ *and* $H(C) = \sum_{a \in C} |a - M_C|$*, in case* M_C *is a single point. In case* M_C *is an interval, then* $H(C) = \sum_{a \in C} |a - b_a|$*, where* $b_a \in M_C$ *and is the nearest point from* a*.*
- C_{1t} *(resp.* C_{2t}*) denotes the positions of the clients that connect to facility* 1 *(resp.* 2*) at stage* t *in the optimal solution* X*.* C_{1t} *(resp.* C_{2t}*) denotes the positions of the clients that connect to facility* 1 *(resp.* 2*) at stage* t *in the solution produced by Algorithm 6.*

With this notation, we are ready to present our algorithm for online 2-Facility Reallocation problem.

4.3.1 The Online Algorithm and a Near Optimal Solution

In this subsection, we present Algorithm 6, which can be seen as a generalization of the *double coverage algorithm* due to the following two reasons. First, it does not necessarily place a facility on the position of the client, since it may connect him (or multiple clients) from a different position. Furthermore, the decisions made by the algorithm have also to take into account the connection cost incurred. Therefore, Step 2 of Algorithm 6 tries to achieve a balance between the moving cost and the connection cost in order to be competitive with the optimal offline solution.

We first mention that Algorithm 6 seems much more complicated than it really is (the first two cases are symmetric both in Step 1 and Step 2). In fact, only the last two cases are

difficult to handle and we explain them subsequently. The performance guarantee of Algorithm 6 is formally stated in Theorem 4.2.

Theorem 4.2. *Let* $x = \{x_1^t, x_2^t\}_{t \quad 1}$ *the solution produced by Algorithm 6 and* x^* *the optimal solution. Then,*

$$Cost(x) \quad 63 \quad Cost(x^*) + |x_1^0 \quad x_1^*| + |x_2^0 \quad x_2^*|,$$

where x_1^0, x_2^0 *are the initial positions of the facilities.*

Algorithm 6 Selecting x_1^t and x_2^t

At stage $t \quad 1$ **the new client positions** $C_t = \{a_1^t, \ldots, a_n^t\}$ **arrive**
Step 1: Moving the facilities towards the clients
$z_1 \quad x_1^{t \quad 1}, z_2 \quad x_2^{t \quad 1}$

- If $z_1 > a_n^t$ *move facility* 1 *to the left until it hits* a_n^t: $z_1 \quad a_n^t$
- If $z_2 < a_1^t$ *move facility* 2 *to the right until it hits* a_1^t: $z_2 \quad a_1^t$
- If $z_1 < a_1^t$ **and** $z_2 > a_n^t$ *move facility* 1 *to the right and facility* 2 *to the left until a facility hits* $[a_1^t, a_n^t]$:
  $z_1 \quad z_1 + \min(|x_1^{t \quad 1} \quad a_1^t|, |x_2^{t \quad 1} \quad a_n^t|)$
  $z_2 \quad z_2 \quad \min(|x_1^{t \quad 1} \quad a_1^t|, |x_2^{t \quad 1} \quad a_n^t|)$

Step 2: Selecting the final position of the facilities

- If $a_1^t \quad z_1 \quad a_n^t$ **and** $z_2 \quad a_n^t \quad 3H(C_t)$:
  put facility 1 *to the median of* C_t: $x_1^t \quad M_{C_t}$
  move facility 2 *to the left by* $3H(C_t)$: $x_2^t \quad z_2 \quad 3H(C_t)$
- If $a_1^t \quad z_2 \quad a_n^t$ **and** $a_1^t \quad z_1 \quad 3H(C_t)$:
  put facility 2 *to the median of* C_t: $x_2^t \quad M_{C_t}$
  move facility 1 *to the right by* $3H(C_t)$: $x_1^t \quad z_1 + 3H(C_t)$
- Else *Compute the partition* (O_1, O_2) *of* C_t *that minimizes the connection cost at stage* t. *Put facility* 1 *to the median of* O_1 *and facility* 2 *to the median of* O_2. $x_1^t \quad M_{O_1}, x_2^t \quad M_{O_2}$

First, we present Lemma 4.3 that is a key component in the subsequent analysis and that reveals the real difficulty of the online 2-Facility Reallocation problem.

Lemma 4.3. *Let the optimal solution be* x^* *and let* C_{1t}, C_{2t} *be the set of clients that connect at stage* t *to facilities* 1,2 *respectively . Let the solution* $y^t = (y_1^t, y_2^t)$ *be defined as follows:*

$$y_k^t = \begin{cases} M_{C_{kt}} & \text{if } C_{kt} \text{ 6= } \text{Æ} \\ x_k^{t} & \text{if } C_{kt} = \text{Æ} \end{cases}.$$

Then, the following inequality holds:

$$\sum_{t=1}^{T} \left[\sum_{k=1}^{2} [H(C_{kt}) + |y_k^t \quad y_k^{t \quad 1}|] \right] \quad 3 \quad Cost(x^*).$$

Proof. Since $\sum_{t=1}^{T}\sum_{k=1}^{2} H(C_{kt}) = \sum_{t=1}^{T}\sum_{k=1}^{2}\sum_{a2C_{kt}} |x_k^t - a|$, we only have to prove that

$$\sum_{t=1}^{T}\sum_{k=1}^{2} |y_k^t - y_k^{t-1}| - 2\sum_{t=1}^{T}\sum_{k=1}^{2}\left[H(C_{kt}) + |x_k^t - x_k^{t-1}|\right]$$

From the triangle inequality, we have that

$$\sum_{t=1}^{T}\sum_{k=1}^{2} |y_k^t - y_k^{t-1}| \quad \sum_{t=1}^{T}\sum_{k=1}^{2}[|y_k^t - x_k^t| + |y_k^{t-1} - x_k^{t-1}| + |x_k^t - x_k^{t-1}|]$$

If $y_k^t = x_k^t$ and $y_k^{t-1} = x_k^{t-1}$, then the right hand side of the inequality is simply the optimal moving cost, which is at most $Cost(x^*)$. If $y_k^t \neq x_k^t$ for $k = 1, 2$, namely when $C_{kt} \neq \emptyset$, then we can bound the quantity $\sum_{t=1}^{T}\sum_{k=1}^{2} |y_k^t - x_k^t|$ by the optimal connection cost. Since y_k^t is the median client (lies in the median interval of C_{kt} in the case $|C_{kt}| = 2k$) of C_{kt} in this case, we have that

$$\sum_{t=1}^{T}\sum_{k=1}^{2} |y_k^t - x_k^t| \quad \sum_{t=1}^{T}\sum_{k=1}^{2}\sum_{a2C_{kt}} |x_k^t - a| \quad \sum_{t=1}^{T}\sum_{k=1}^{2} H(C_{kt}).$$

Since the same arguments hold in the case $y_k^{t-1} \neq x_k^{t-1}$ for $k = 1, 2$, we have that:

$$\sum_{t=1}^{T}\sum_{k=1}^{2}[|y_k^t - x_k^t| + |y_k^{t-1} - x_k^{t-1}| \quad 2\sum_{t=1}^{T}\sum_{k=1}^{2} H(C_{kt}).$$

□

Lemma 4.3 indicates that the real difficulty of the problem is not determining the exact positions of the facilities in the optimal solution, but to determine the *service clusters* that the optimal solution forms. In fact, if we knew the clusters C_{1t}, C_{2t}, then Lemma 4.3 provides us with a 3-approximation algorithm. Obviously, this information cannot be acquired in the online setting, since C_{1t}, C_{2t} depend on the future positions of the clients that we do not know. We prove that Algorithm 6 has an approximation guarantee of 21 with respect to the solution y, that directly translates to an approximation guarantee of 63 with respect to $Cost(x^*)$. The latter is formally stated in Lemma 4.4 and is the main result of this section.

Lemma 4.4. *Let $x = \{x_1^t, x_2^t\}_{t-1}$ be the solution produced by Algorithm 6. Then, the cost paid by solution x at stage t, $\sum_{k=1}^{2} |x_k^t - x_k^{t-1}| + \sum_{i=1}^{n} \min_{k=1,2} |x_k^t - a_i^t|$, is at most*

$$21\sum_{k=1}^{2}[H(C_{kt}) + |y_k^t - y_k^{t-1}|] + F_t(x^t) - F_{t-1}(x^{t-1}),$$

where $F_t(x_1, x_2) = 2(|x_1 - y_1^t| + |x_2 - y_2^t|) + |x_1 - x_2|$.

Lemma 4.4 directly implies Theorem 4.2 by applying a telescopic sum over all t and then applying Lemma 4.3. Notice that the additive term $|x_1^0 - x_2^0|$ in Theorem 4.2 depends only on the initial positions of the facilities and follows from the fact that $F_0(x^0) = |x_1^0 - x_2^0|$. Note that the additive term is a constant independent from the request sequence (the client positions C_t). As the request sequence grows, the additive term becomes negligible, therefore it is common to define the competitive ratio of an online algorithm in such a way [59, 55].

4.3.2 Bounding the Cost of the Online Algorithm

In this subsection, we present the proof ideas of Lemma 4.4, which come together with explaining Steps 1 and 2 of our algorithm. Let us start with explaining Step 1.

We remind that Step 1 is performed by Algorithm 6 if both facilities are outside the interval C_t at the beginning of stage t. Then, either both facilities are on the same side of C_t or one of them is on the left and the other on the right. Therefore, the online algorithm distinguishes between the three cases depicted in Figure 4.1 (We show 2 cases since the case with both facilities on the right of the clients is symmetric to the first). Notice that moving with the same speed towards the interval $[a_1^t, a_n^t]$ results to the same moving cost for both facilities; both facilities will move the distance of the facility which is closest to its closest client.

The following lemma bounds the online cost paid after the execution of Step 1. First, note that since $x_1^0 \leq x_2^0$, then $x_1^t \leq x_2^t$ by our algorithm construction. Now, assume that $x_2^{t-1} < a_1^t$ (second case). Before deciding the exact positions of the facilities, we can *safely* move facility 2 to the right until reaching a_1^t. The term *safely* means that this moving cost is *roughly* upper bounded by the moving cost $\sum_{k=1}^{2} |y_k^t - y_k^{t-1}|$. This *safe moving* applies to all three cases of Step 1 in Algorithm 6 and is formally stated in Lemma 4.5.

Lemma 4.5. *Let $z = (z_1, z_2)$ denote the values of the variables z_1, z_2 after Step 1 of Algorithm 6. Then,*

$$\sum_{k=1}^{2} |z_k - x_k^{t-1}| - 2 \sum_{k=1}^{2} |y_k^t - y_k^{t-1}| \leq F_t(z) + F_{t-1}(x^{t-1}).$$

Proof. Assume that $x_2^{t-1} < a_1^t$, then Algorithm 6 will first move facility 2 to a_1^t ($z_1 = x_1^{t-1}, z_2 = a_1^t$), paying moving cost equal to $|a_1^t - x_2^{t-1}|$. This moving cost can be bounded

with the use of the potential function Φ. More specifically, we have that:

$$\begin{aligned}\Phi_t(z) - \Phi_{t-1}(x^{t-1}) &= \Phi_t(z) - \Phi_t(x^{t-1}) + \Phi_t(x^{t-1}) - \Phi_{t-1}(x^{t-1}) \\ &= \Phi_t(z) - \Phi_t(x^{t-1}) + 2\sum_{k=1}^{2}(|y_k^t - x_k^{t-1}| - |y_k^{t-1} - x_k^{t-1}|) \\ &\leq \Phi_t(z) - \Phi_t(x^{t-1}) + 2\sum_{k=1}^{2}|y_k^t - y_k^{t-1}|. \end{aligned} \tag{4.1}$$

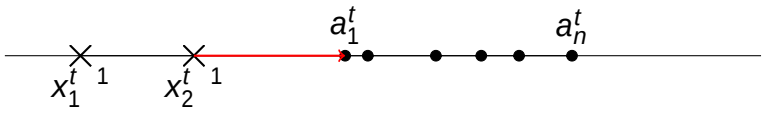
In the considered case $z_1 = x_1^{t-1}, z_2 = a_1^t$, the difference $\Phi_t(z) - \Phi_t(x^{t-1})$ in the potential function equals the quantity $2(|y_2^t - a_1^t| - |y_2^t - x_2^{t-1}|) + |x_1^{t-1} - a_1^t| - |x_1^{t-1} - x_2^{t-1}|$. By the definition of solution y in Lemma 4.4, either y_1^t or y_2^t lies in the interval $[a_1^t, a_n^t]$. Since either y_1^t or y_2^t lies in the interval $[a_1^t, a_2^t]$ and $y_1^t \leq y_2^t$, we have that $a_1^t \leq y_2^t$. Meaning that z_2 is closer to y_2^t than x_2^{t-1} and consequently $2(|y_2^t - a_1^t| - |y_2^t - x_2^{t-1}|) = 2|x_2^{t-1} - a_1^t|$. Therefore, $\Phi_t(z) - \Phi_t(x^{t-1}) = 2|x_2^{t-1} - a_1^t| + |x_1^{t-1} - a_1^t| - |x_1^{t-1} - x_2^{t-1}| = |a_1^t - x_2^{t-1}| = |z_2 - x_2^{t-1}|$, which completes the proof of Lemma 4.5 for this case of Step 1.

Notice that inequality (4.1) holds for all three cases of Step 1. Thus, one just need to prove that $\Phi_t(z) - \Phi_t(x^{t-1}) \leq \sum_{k=1}^{2}|z_k - x_k^{t-1}|$ for the other two cases. We prove it for the third case of Step 1, since the second case ($x_1^{t-1} \geq a_n^t$) is just symmetric to the first case.

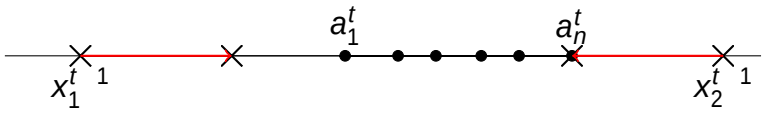
In the third case of Step 1, we have that $x_1^{t-1} < a_1^t, x_2^{t-1} > a_n^t, z_1 = x_1^{t-1} + \min(|x_1^{t-1} - a_1^t|, |x_2^{t-1} - a_n^t|)$ and $z_2 = x_2^{t-1} - \min(|x_1^{t-1} - a_1^t|, |x_2^{t-1} - a_n^t|)$. The difference $\Phi_t(z) - \Phi_t(x^{t-1})$ in the potential function equals the quantity $2(|z_1 - y_1^t| - |x_1^{t-1} - y_1^t| + |z_2 - y_2^t| - |x_2^{t-1} - y_2^t|) + |z_1 - z_2| - |x_1^{t-1} - x_2^{t-1}|$. Now, $|z_1 - z_2| - |x_1^{t-1} - x_2^{t-1}| = 2\min(|x_1^{t-1} - a_1^t|, |x_2^{t-1} - a_n^t|) = \sum_{k=1}^{2}|z_k - x_k^{t-1}|$. Assume that $y_1^t \in [a_1^t, a_n^t]$, then $\sum_{k=1}^{2}(|z_k - y_k^t| - |x_k^{t-1} - y_k^t|) \leq 0$ since $|z_1 - y_1^t| - |x_1^{t-1} - y_1^t| = \min(|x_1^{t-1} - a_1^t|, |x_2^{t-1} - a_n^t|)$ and $|z_2 - y_2^t| - |x_2^{t-1} - y_2^t| \leq \min(|x_1^{t-1} - a_1^t|, |x_2^{t-1} - a_n^t|)$. As a result, inequality (4.1) holds. Using the same argument in case $y_2^t \in [a_1^t, a_n^t]$ completes the proof. □

The proof of Lemma 4.5 reveals why we compare our algorithm with the solution y and not directly with x. All these *safe moves* are based on the fact that either y_1^t or y_2^t lies in the $C_t = [a_1^t, a_n^t]$ (the latter does not necessarily hold for x). Finally, the *potential function* $\Phi_t(x_1, x_2)$ is crucial, since it permits *safe moves*, when all clients are on the right/left of the facilities (first/second case) as well as when they are contained in the interval $[x_1^{t-1}, x_2^{t-1}]$ (third case).

It is clear, that any reasonable algorithm will move at least one facility inside C_t in order to serve the clients. Lemma 4.5 shows that this moving cost can be charged to the difference $\Phi_t(x^{t-1}) - \Phi_t(z)$ in the potential function. Now, we will show that we can charge the cost of the second step of Algorithm 6 to the difference $\Phi_t(x^t) - \Phi_t(z)$. In Step 2, we need to bound the connection cost plus some additional moving cost from the point where *the*



If both facilities 1 and 2 are on the left of the clients, then facility 2 is moved to the right until hitting the position of the leftmost client (the case with facilities 1 and 2 on the right of clients is symmetric).



If facility 1 is on the left of the clients and facility 2 is on the right of the clients, then both facilities are moved with the same speed towards the interval $[a_1^t, a_n^t]$ until one of them hits the interval.

FIGURE 4.1: Step 1 of Algorithm 6 is depicted. After this step, the positions of the facilities are denoted by z_1, z_2 in Algorithm 6.

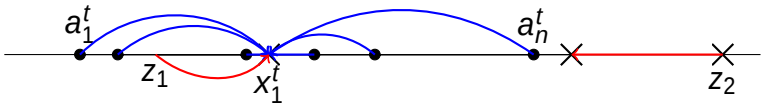
safe move stopped. This is the step, where the final positions of the facilities are determined and also the most challenging one in the analysis, since Algorithm 6 has to decide whether it will serve the clients using both facilities (one of them will definitely serve some of the clients) without knowing what the optimal solution does.

Before we prove the guarantees of Step 2, we will explain the cases exhibited in this step in high level (see Figure 4.2). In the first case, where the second facility is close to C_t, the online algorithm serves the clients using both facilities. Then, Algorithm 6 will pay optimal connection cost and small moving cost, since the first facility is already inside C_t and the second is close to C_t. In the second case, where the second facility is far from C_t, the clients are served with one facility and the second facility is moved by an appropriate distance towards C_t. In this case, the optimal connection cost can be arbitrarily smaller than the connection cost of our online algorithm. However, moving the second facility by an appropriate distance decreases $F_t(x^t)$ $F_t(z)$ so as to cancel the cost incurred by the online algorithm.

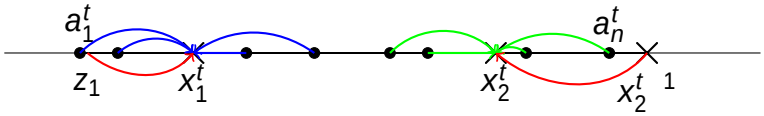
We are now ready to prove Lemma 4.6, which formalizes the guarantees provided by Algorithm 6 after the execution of Step 2. Algorithm 6 keeps a balance between the moving cost of the facilities and the connection cost in order to be competitive with the optimal solution as will become apparent from the analysis.

Lemma 4.6. Let $x^t = (x_1^t, x_2^t)$ denote the locations of facilities at stage t after the execution of Step 2. Then,

$$\sum_{k=1}^{2} [H(C_{kt}) + |x_k^t - z_k|] \leq 21 \sum_{k=1}^{2} H(C_{kt}) + F_t(x^t) + F_t(z).$$



The first choice of Step 2 is depicted. In this case, the facility initially lying inside the interval $[a_1^t, a_n^t]$ moves to the median of clients. In this position, the connection cost is minimized using one facility.



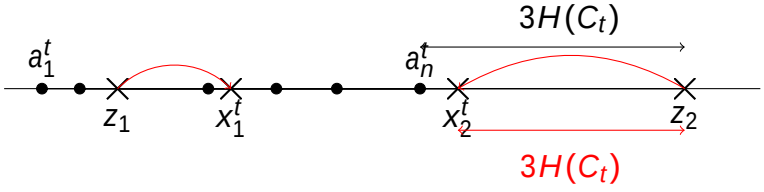
The second choice of Step 2 is depicted. Facilities are placed to the positions, where the connection cost of the clients is minimized using two facilities.

FIGURE 4.2: Step 2 of Algorithm 6 is depicted.

Proof. Observe that by Algorithm 6, either a_1^t z_1 a_n^t or a_1^t z_2 a_n^t. As a result, we need to prove the claim for the following 4 cases:

- a_1 z_1 a_n **and** z_2 a_n $3H(C_t)$
- a_1 z_1 a_n **and** z_2 a_n $< 3H(C_t)$
- a_1 z_2 a_n **and** a_1 z_1 $3H(C_t)$
- a_1 z_2 a_n **and** a_1 z_1 $< 3H(C_t)$

We will prove just the first and the second case since the third is symmetric to the first and the forth is symmetric to the second. In case a_1 z_1 a_n and z_2 a_n $3H(C_t)$, Algorithm 6 puts facility 1 in the median of C_t, namely $x_1^t = M_{C_t}$ (or x_1^t 2 M_{C_t} in case the number of clients is even), and moves facility 2 to the left by a distance of $3H(C_t)$ as can be seen below.



First note that $\sum_{k=1}^{2} H(C_{kt})$ $H(C_t)$ since x_1^t 2 M_{C_t}. Then jx_1^t $z_1 j$ ja_1^t $a_n^t j$ $H(C_t)$ because both x_1^t and z_1 lie in the interval $[a_1^t, a_n^t]$ and jx_2^t $z_2 j = 3H(C_t)$ by Algorithm 6. Therefore, we have that the cost of the online algorithm is $\sum_{k=1}^{2} H(C_{kt}) + jx_k^t$

$z_k j \quad 5H(C_t)$. By the geometry of this case and the aforementioned bounds,

$$\mathrm{F}_t(x^t) \quad \mathrm{F}_t(z) = 2 \sum_{k=1}^{2} (j x_k^t \quad y_k^t j \quad j z_k \quad y_k^t j) + j x_1^t \quad x_2^t j \quad j z_1 \quad z_2 j$$
$$2 \sum_{k=1}^{2} (j x_k^t \quad y_k^t j \quad j z_k \quad y_k^t j) \quad 2H(C_t).$$

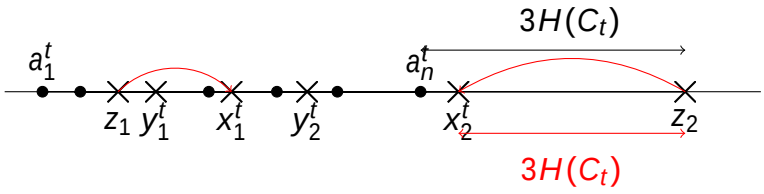
Since

$$\sum_{k=1}^{2} [H(C_{kt}) + j x_k^t \quad z_k j] \quad 5H(C_t),$$

bounding the term

$$\sum_{k=1}^{2} (j x_k^t \quad y_k^t j \quad j z_k \quad y_k^t j)$$

by the optimal connection cost $\sum_{k=1}^{2} H(C_{kt})$ is now the challenge. The real difficulty arises when C_{1t} ~~6=~~ Æ and C_{2t} ~~6=~~ Æ, where $\sum_{k=1}^{2} H(C_{kt})$ can be arbitrarily smaller than $H(C_t)$. As we will see in this case (see also the figure below) x^t gets closer to y^t and the term $\sum_{k=1}^{2} (j x_k^t \quad y_k^t j \quad j z_k \quad y_k^t j)$ becomes negative.



Since C_{2t} ~~6=~~ Æ and y_2^t 2 $M_{C_{2t}}$ we get that $y_2^t \quad a_n^t$ and as a result $j x_2^t \quad y_2^t j \quad j z_2 \quad y_2^t j = j x_2^t \quad z_2 j \quad 3H(C_t)$.

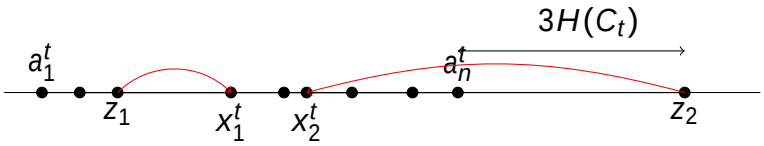
$$\mathrm{F}_t(x^t) \quad \mathrm{F}_t(z) \quad 2 \sum_{k=1}^{2} (j x_k^t \quad y_k^t j \quad j z_k \quad y_k^t j) \quad 2H(C_t)$$
$$= 2 (j x_1^t \quad y_1^t j \quad j z_1 \quad y_1^t j) + 2 (j x_2^t \quad z_2 j \quad j z_2 \quad y_2^t j) \quad 2H(C_t)$$
$$2 j x_1^t \quad z_1 j \quad 8H(C_t)$$
$$2H(C_t) \quad 8H(C_t)$$
$$6H(C_t)$$
$$\sum_{k=1}^{2} H(C_{kt}) \quad \sum_{k=1}^{2} [H(C_{kt}) + j x_k^t \quad z_k j].$$

Now, assume that $C_{1t} = \text{Æ}$ or $C_{2t} = \text{Æ}$ meaning that $\sum_{k=1}^{2} H(C_{kt}) = H(C_t)$. As a result, bounding *everything* by $H(C_t)$ serves our purpose. More formally,

$$\begin{aligned} F_t(x^t) \quad F_t(z) \quad & 2 \sum_{k=1}^{2} \left(|x_k^t \quad y_k^t| \quad |z_k \quad y_k^t| \right) \quad 2H(C_t) \\ & 2 \sum_{k=1}^{2} |x_k^t \quad z_k| \quad 2H(C_t) \\ & 6H(C_t) \\ & 11H(C_t) \quad \sum_{k=1}^{2} \left[H(C_{kt}) + |x_k^t \quad z_k| \right] \\ = 11 \sum_{k=1}^{2} H(C_{kt}) \quad & \sum_{k=1}^{2} \left[H(C_{kt}) + |x_k^t \quad z_k| \right]. \end{aligned}$$

The fourth inequality follows from the fact that $\sum_{k=1}^{2} H(C_{kt}) + |x_k^t \quad z_k| \quad 5H(C_t)$.

We now need to treat the second case where $a_1 \quad z_1 \quad a_n$ and $z_2 \quad a_n < 3H(C_t)$. Since Algorithm 6 computes the optimal clustering (C_{1t}, C_{2t}) and puts x_1^t in the interval $M_{C_{1t}}$ and x_2^t in the interval $M_{C_{2t}}$, we are ensured that the connection cost of our solution is less than the connection cost of y^t, $\sum_{k=1}^{2} H(C_{kt}) \quad \sum_{k=1}^{2} H(C_{kt})$, so we are mostly concerned in bounding $\sum_{k=1}^{2} |x_k^t \quad z_k|$.



The easy case is when $\sum_{k=1}^{2} H(C_{kt}) = H(C_t)$. A small difference with the previous case is that we don't know how $|x_2^t \quad z_2|$ is. However, $z_1, x_1^t, x_2^t \; 2 \; [a_1^t, \ldots a_n^t]$ and $|x_2^t \quad z_2| = |x_2^t \quad a_n^t| + |a_n^t \quad z_2|$. Thus, $|x_1^t \quad z_1| + |x_2^t \quad a_n^t| \quad H(C_t)$, $|a_n^t \quad z_2| \quad 3H(C_t)$ and therefore $\sum_{k=1}^{2} [H(C_{kt}) + |x_k^t \quad z_k|] \quad 5H(C_t)$. So we can again bound *everything* by $H(C_t)$.

$$\begin{aligned}
\Phi_t(x^t) \quad \Phi_t(z) &= 2\sum_{k=1}^{2}\left(|x_k^t \quad y_k^t| + |z_k \quad y_k^t|\right) + |x_1^t \quad x_2^t| \quad |z_1 \quad z_2| \\
&\quad 3\sum_{k=1}^{2}|x_k^t \quad z_k| \\
&\quad 4\sum_{k=1}^{2}|x_k^t \quad z_k| \quad \sum_{k=1}^{2}|x_k^t \quad z_k| \\
&\quad 20H(C_t) \quad \sum_{k=1}^{2}\left[|x_k^t \quad z_k|\right] \\
&\quad 21\sum_{k=1}^{2}H(C_{kt}) \quad \sum_{k=1}^{2}\left[H(C_{kt}) + |x_k^t \quad z_k|\right].
\end{aligned}$$

Things become more complicated, when the connection cost $\sum_{k=1}^{2} H(C_{kt})$ is relatively small ($C_{1t} \leq \text{Æ}$ and $C_{2t} \leq \text{Æ}$), where bounding everything by $H(C_t)$ does not work. However, the solutions x^t and y^t will be relatively close in this case. More formally,

$$\begin{aligned}
\Phi_t(x^t) \quad \Phi_t(z) &= 2\sum_{k=1}^{2}\left(|x_k^t \quad y_k^t| + |z_k \quad y_k^t|\right) + |x_1^t \quad x_2^t| \quad |z_1 \quad z_2| \\
&= 2\sum_{k=1}^{2}|x_k^t \quad y_k^t| \quad 2\sum_{k=1}^{2}|z_k \quad y_k^t| + \sum_{k=1}^{2}|x_k^t \quad z_k| \\
&= 2\sum_{k=1}^{2}|x_k^t \quad y_k^t| + 2\sum_{k=1}^{2}\left(|x_k^t \quad z_k| + |z_k \quad y_k^t|\right) \quad \sum_{k=1}^{2}|x_k^t \quad z_k| \\
&\quad 4\sum_{k=1}^{2}|x_k^t \quad y_k^t| \quad \sum_{k=1}^{2}|x_k^t \quad z_k|.
\end{aligned}$$

We need to upper bound the distance $\sum_{k=1}^{2}|x_k^t \quad y_k^t|$. Observe that in the solution x^t, the client at position a_1^t connects to the left facility (facility 1) and the client at position a_n^t connects to the right facility (facility 2), $|x_1^t \quad a_1^t| + |x_2^t \quad a_n^t| \quad \sum_{k=1}^{2} H(C_{kt})$. Since $C_{1t} \leq \text{Æ}$ and $C_{2t} \leq \text{Æ}$, the same holds for the solution y^t. As a result,

$$\begin{aligned}
\Phi_t(x^t) \quad \Phi_t(z) \quad & 4\sum_{k=1}^{2}|x_k^t \quad y_k^t| \quad \sum_{k=1}^{2}|x_k^t \quad z_k| \\
& 4\left(|x_1^t \quad a_1^t| + |y_1^t \quad a_1^t| + |x_2^t \quad a_n^t| + |y_2^t \quad a_n^t|\right) \quad \sum_{k=1}^{2}|x_k^t \quad z_k| \\
& 4\sum_{k=1}^{2}\left[H(C_{kt}) + H(C_{kt})\right] \quad \sum_{k=1}^{2}|x_k^t \quad z_k| \\
& 9\sum_{k=1}^{2}H(C_{kt}) \quad \sum_{k=1}^{2}\left[H(C_{kt}) + |x_k^t \quad z_k|\right].
\end{aligned}$$

□

This completes the proof of the performance of our online algorithm Algorithm 6 and concludes this section.

5 Competitive Algorithms for the Online Dynamic Facility Location Problem

5.1 Introduction

In this chapter, we study the online variant of the Dynamic Facility Location problem. We present a randomized $O(\log m + \log n)$-competitive algorithm, where m is the number of facilities and n is the number of clients. The algorithm produces a fractional solution, in each timestep, to the objective of Dynamic Facility Location involving a regularization function. Then, it rounds the fractional solution of this timestep to an integral one with the use of exponential clocks. We complement our result by proving a lower bound of $\Omega(m)$ for deterministic algorithms and lower bound of $\Omega(\log m)$ for randomized algorithms. These lower bound results are summarized in the following theorem.

Theorem 5.1. *The competitive ratio of any deterministic online algorithm is* $\Omega(m)$ *and the competitive ratio of any randomized online algorithm against the oblivious adversary is* $\Omega(\log m)$ *for the Online Dynamic Facility Location problem, where* m *is the number of facilities.*

Our first step to design a competitive algorithm for ODFL is to prove that ODFL fits in the framework of [26]. This framework provides a general algorithm for solving online problems, which satisfy certain types of time varying constraints, by using a regularization technique from online learning that can also produce competitive solutions for dynamic online problems (see also [27]). This seemed intractable using standard competitive analysis methods, despite the significant differences between the fields of competitive analysis and online learning.

In order to design the randomized competitive algorithm , we first express the offline Dynamic Facility Location problem as a linear program P (Figure 5.1a). Then, we apply the following two algorithms at each round t.

1. **Algorithm 7 (Regularization algorithm)**: It solves a linear program minimizing the objective function of P modified to include a smooth convex regularization term and obtains the fractional solution $Sol(t)$.

2. **Algorithm 8 (Rounding algorithm)**: It rounds the fractional solution $Sol(t)$ of Algorithm 7 to an integral solution using competing exponential clocks .

Algorithm 7 solves online a linear program to produce a fractional solution at each round t involving the current distance vector d_t. This algorithm is essentially the general algorithm presented in [26], which we adapt to ODFL. The performance of the general regularization algorithm is proved by Theorem 1.1 in [26] for the case of time varying covering constraints. Although we follow the same steps to prove the existence of a $O(\log m)$-competitive fractional solution for ODFL, where m is the number of facilities, we must also address the presence of both covering and precedence constraints in ODFL.

Algorithm 8 is the randomized procedure that rounds the fractional solution provided from Algorithm 7 to an integral solution. Our contribution here is that we use an appropriate rounding which works favorably with Algorithm 7 so as to produce a solution, which is $O(\log m + \log n)$-competitive for ODFL. The rounding algorithm makes use of competing exponential clocks, which have been applied in many similar problems like the Dynamic Facility Location problem [4] and the Online Set Cover with Service Cost problem [26]. Combining Algorithm 7 and Algorithm 8 we get the following result.

Theorem 5.2. *There is a randomized algorithm which is $O(\log m + \log n)$-competitive for the Online Dynamic Facility Location problem, where m denotes the number of facility locations and n denotes the number of clients.*

5.2 Lower Bounds

In this section, we prove lower bounds of deterministic and randomized algorithms for ODFL. In both cases, the metric space is a star graph with a client lying on the center of the star for all rounds.

The core idea of the proofs is to force the online algorithm to pay the switching cost at each round. By carefully selecting the parameters of ODFL, we can prove that any deterministic online algorithm is $O(m)$-competitive. For the randomized lower bound we use Yao's principle (see examples in Chapter 8 in [23]). Specifically, we choose a randomized instance such that the expected performance of any deterministic algorithm against the optimal offline algorithm is $\Omega(\log m)$. By Yao's principle, any randomized algorithm has the same lower bound.

Theorem 5.1. *The competitive ratio of any deterministic online algorithm is $\Omega(m)$ and the competitive ratio of any randomized online algorithm against the oblivious adversary is $\Omega(\log m)$ for the Online Dynamic Facility Location problem, where m is the number of facilities.*

Proof. Let OPT denote the optimal cost and ALG denote the cost of an online algorithm. The instance consists of a star graph with m edges and the number of rounds is $T = m$. Facilities can only be opened in the leaves (a total of m leaves) and there is one client ($n = 1$) sitting at the center of a graph for all rounds. The distance of every leaf j to the center is initially $d_j = d$. Then, the adversary has the following simple strategy at each round $1 \quad t \quad T \quad 1$:

For every leaf j, such that the online algorithm connects the client to the facility in j, d_j becomes arbitrarily large. At round T the distances remain the same as in the previous round.

Observe that there is only one leaf with distance d from the center of a star for all rounds. The optimal offline solution just opens a facility at this leaf and connects the client to it for all rounds, thus pays $g + Tf + Td$. On the other side, any competitive online algorithm will prefer to open a new facility at distance d and connect the client to this facility at the start of each round instead of paying the large distance. Therefore, the cost incurred by any online algorithm is at least $Tg + Tf + Td$. By setting $g \quad d = f$, we have that

$$\frac{\text{ALG}}{\text{OPT}} \quad \frac{Tg + Tf + Td}{g + Tf + Td} = \Omega(T) = \Omega(m)$$

Turning to the randomized case, the instance consists of the same metric as the deterministic case and the only difference is that we will use randomized adversarial requests. Then, by showing that any deterministic algorithm has competitive ratio at least $\log m$ and by Yao's principle, we will prove the lower bound for randomized algorithms.

Now, the adversary chooses uniformly at random an edge e, which has length d (has not yet become arbitrarily large) at each round $1 \quad t \quad T \quad 1$ and makes its length arbitrarily large. At round T, where only one leaf has distance d, the distances remain the same as in the previous round. Again, the optimal solution uses the leaf in distance d at all rounds and pays $g + Tf + Td$. However, the expected switching cost of any competitive algorithm is:

$$\mathbb{E}[\text{switching cost}] = g + \sum_{t=1}^{T \quad 1} \Pr[\text{switches at round } t] \quad g = g + \sum_{t=1}^{T \quad 1} \frac{1}{m \quad t + 1} \quad g > H_T \quad g$$

since at each round t the edge that the algorithm uses becomes arbitrarily large with probability $1/(m \quad t + 1)$. By setting $g > Td = Tf$,

$$\frac{\mathbb{E}[\text{ALG}]}{\text{OPT}} \quad \frac{g \quad H_T + Tf + Td}{g + Tf + Td} = \Omega(\log T) = \Omega(\log m).$$

□

This concludes this section with the lower bounds. In the following sections, we present a randomized algorithm for the ODFL problem with a nearly matching bound of $O(\log m + \log n)$.

5.3 The Regularization Algorithm

In this section, we show that the regularization algorithm of [26] can be applied to ODFL and that it produces a fractional solution at each round, which is $O(\log m)$-competitive, where m is the number of facility locations. We will prove the following theorem:

Theorem 5.3. *The Regularization Algorithm produces an $O(\log m)$-competitive fractional solution for the Online Dynamic Facility Location problem, where m is the number of facilities.*

Before proceeding to the details of Algorithm 7, we first express the offline Dynamic Facility Location as a linear program, denoted as P (Figure 5.1a). Algorithm 7 will solve a linear program P at each round, which will be constructed from P combined with a regularization function. Finally, we will show that the fractional solution of P is $O(\log m)$-competitive with respect to the solution of the dual program D (Figure 5.1b) of P, which serves as lower bound on the optimal offline solution.

Now, we express offline Dynamic Facility Location as an integer program, which will be relaxed to obtain the linear program P. Recall that T, n, m are the number of rounds, clients and facility locations, respectively. The first term of the objective function is the total facility opening cost, where f is the cost to open a facility. The second term is the total connection cost, where $d_t(i, j)$ is the distance between facility i and client j in round t, and the third term is the total switching cost, where each change of a client's connection to a facility costs g.

We use the decision variables y_i^t, x_{ij}^t and z_{ij}^t, where $i \in [m], j \in [n], t \in [T]$; $y_i^t = 1$ if facility i is open at round t and $y_i^t = 0$ otherwise, $x_{ij}^t = 1$ if client j is connected to facility i at round t and $x_{ij}^t = 0$ otherwise, $z_{ij}^t = 1$ if client j was connected to facility i at round t but not connected to the same facility i at round $t - 1$ and $z_{ij}^t = 0$ otherwise. The value of the variable z_{ij}^t is imposed from the third constraint, which expresses the switching cost. The first constraint ($x_{ij}^t - y_i^t$) ensures that whenever a client j is connected to a facility i, the facility i is open. The second constraint ($\sum_{i=1}^{m} x_{ij} - 1$) guarantees that every client is connected to a facility. Finally, relaxing the decision variables to take non-negative real values we obtain the LP of Figure 5.1a, denoted as P.

Next, we are ready to present Algorithm 7. The algorithm is given at each round t a distance vector $d_t \in \mathbb{R}_+^{m \times n}$ containing the distances between clients and facilities. Then, Algorithm 7 finds the minimizer (y^t, x^t) of the linear program P at each round t, which has two differences from P. The first one is that the last term of the objective function in P (the switching

$$
\begin{array}{lll}
\min & f \sum_{t=1}^{T} \sum_{i=1}^{m} y_i^t + \sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij}^t \; d_t(i,j) + g \sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{i=1}^{m} z_{ij}^t & \\
\text{s.t.} & x_{ij}^t \quad y_i^t & \forall t \in [T], \forall i \in [m], \forall j \in [n] \\
& \sum_{i=1}^{m} x_{ij}^t \quad 1 & \forall t \in [T], \forall j \in [n] \\
& z_{ij}^t \quad x_{ij}^t \quad x_{ij}^{t-1} & \forall t \in [T], \forall i \in [m], \forall j \in [n] \\
& y_i^t \quad 0, x_{ij}^t \quad 0, z_{ij}^t \quad 0 & \forall t \in [T], \forall i \in [m], \forall j \in [n]
\end{array}
$$

(A) The linear program P for offline Dynamic Facility Location.

$$
\begin{array}{lll}
\max & \sum_{t=1}^{T} \sum_{j=1}^{n} a_j^t & \\
\text{s.t.} & \sum_{j=1}^{n} e_{ij}^t \quad f & \forall t \in [T], \forall i \in [m] \\
& b_{ij}^t \quad g & \forall t \in [T], \forall i \in [m], \forall j \in [n] \\
& b_{ij}^{t+1} \quad b_{ij}^t \quad d_t(i,j) + e_{ij}^t \quad a_j^t & \forall t \in [T], \forall i \in [m], \forall j \in [n] \\
& b_{ij}^t \quad 0, e_{ij}^t \quad 0, a_j^t \quad 0 & \forall t \in [T], \forall i \in [m], \forall j \in [n]
\end{array}
$$

(B) The dual D of the linear program P.

cost) is substituted by the regularization function in P . The second is that the constraint relative to the switching cost ($z_{ij}^t \quad x_{ij}^t \quad x_{ij}^{t-1}$) in P is omitted in P . We remark that the regularized objective function includes both the previous solution as well as the current cost vector. Thus, the solution in each round is determined greedily and independently of rounds prior to $t \quad 1$.

To analyze the performance of Algorithm 7 we will need to construct a lower bound on the optimal offline solution. Therefore, we derive the dual D of P (Figure 5.1b), which has the following variables (corresponding to the primal constraints on the left):

- $x_{ij}^t \quad y_i^t \; ! \; e_{ij}^t$ for all $t \in [T], i \in [m], j \in [n]$
- $\sum_{i=1}^{m} x_{ij}^t \quad 1 \; ! \; a_j^t$ for all $t \in [T], j \in [n]$
- $z_{ij}^t \quad x_{ij}^t \quad x_{ij}^{t-1} \; ! \; b_{ij}^t$ for all $t \in [T], i \in [m], j \in [n]$

We will prove Theorem 5.3 by showing that the set of dual variables of the solutions that P returns is a feasible solution for D within a factor of $(1 + (1 + e^{\emptyset}) \ln(1 + \frac{m}{e^{\emptyset}}))$ of the optimal offline solution, where $e^{\emptyset}$ is a small constant. Specifically, we will use the KKT optimality conditions of P (the regularized LP) in each round. The constraints define dual variables, which will be plugged in the formulation of the dual D in Figure 5.1b. This way we will construct a dual solution to the original online problem, which will serve as a lower bound on the optimal offline solution.

[26] remarks that their technique can be generalized to facility location problems, without

Algorithm 7 The regularization algorithm

Parameters: $e > 0, h = \ln(1 + n/e)$
Initialization: Set $y_i^0 = 0 \; 8i \; 2 \; [m]$ and $x_{ij}^0 = 0 \; 8i \; 2 \; [m], j \; 2 \; [n]$.
At each round t: Let $d_t \; 2 \; \mathrm{R}_+^{m \quad n}$ be the distance cost vector and let S be the set of feasible solutions. Solve the following linear program (P) to obtain the fractional solution (y^t, x^t):

$$(y^t, x^t) = \arg\min_{(y,x)2S} \left\{ f \sum_{i=1}^{m} y_i + \sum_{j=1}^{n} \sum_{i=1}^{m} x_{ij} \quad d_t(i,j) + \frac{1}{h} \sum_{j=1}^{n} \sum_{i=1}^{m} \left[\left(\left(x_{ij} + \frac{e}{n} \right) \ln \frac{x_{ij} + \frac{e}{n}}{x_{ij}^{t \; 1} + \frac{e}{n}} \right) \quad x_{ij} \right] \right\}$$

providing any further technical details. In the next lemmas, we verify their claim, by adjusting their approach and proof techniques to ODFL. Recall that the constraint $z_{ij}^t \quad x_{ij}^t \quad x_{ij}^{t \; 1}$ is omitted in P . In order to define a feasible solution for D, we introduce the variable b_{ij}^t corresponding to this constraint and we let e_{ij}, a_j be the optimal dual variables of D corresponding to the precedence and covering constraints respectively.

Lemma 5.1. *The set of optimal solutions for each round t of the dual LP D of P $(a^{,t}, e^{,t})$, which satisfy the KKT conditions for an appropriate b_{ij}^t, consist of a feasible solution for D.*

Proof. Let $x^{,t}$ be the optimal solution of P at round t. Set the variables of D at time t to be:

$$a_j^t = a_j^{,t}, \; e_{ij}^t = e_{ij}^{,t} \text{ and } b_{ij}^{t+1} = \frac{g}{h} \ln \frac{1 + \frac{e}{n}}{x_{ij}^{,t} + \frac{e}{n}}$$

To prove that the solution above is feasible for D, we prove that it satisfies its constraints one by one. This is achieved using the following KKT conditions that hold for P and its dual:

$$a_j \quad 0 \; , 8j \; 2 \; [n] \tag{5.1}$$

$$e_{ij} \quad 0 \; , 8i \; 2 \; [m], 8j \; 2 \; [n] \tag{5.2}$$

$$f \quad \sum_{j=1}^{n} e_{ij} \quad 0 \; , 8i \; 2 \; [m] \tag{5.3}$$

$$d_t(i,j) + \frac{g}{h} \ln \frac{x_{ij} + \frac{e}{n}}{x_{ij}^{t \; 1} + \frac{e}{n}} + e_{ij} \quad a_j \quad 0 \; , 8i \; 2 \; [m], 8j \; 2 \; [n] \tag{5.4}$$

The first group of constraints of the dual D (Figure 5.1b) ($\sum_{j=1}^{n} e_{ij}^t \quad f$) follows easily from KKT condition (3). The same holds for the last two groups of constraints ($e_{ij}^t \quad 0$ and $a_j^t \quad 0$)

due to KKT conditions (1) and (2). Furthermore, by (4) and the construction of b_{ij}^t we have that:

- $b_{ij}^t = \frac{g}{h} \ln \frac{1+\frac{e}{n}}{x_{ij}^{t-1}+\frac{e}{n}} = \frac{g}{\ln(1+\frac{n}{e})} \ln \frac{1+\frac{e}{n}}{x_{ij}^{t-1}+\frac{e}{n}} \quad x^t \quad 0 \quad \frac{g}{\ln(1+\frac{n}{e})} \ln \frac{1+\frac{e}{n}}{\frac{e}{n}} = \frac{g}{\ln(\frac{n}{e}+1)} \ln(\frac{n}{e} + 1) = g$
- $b_{ij}^{t+1} \quad b_{ij}^t = \frac{g}{h} \ln \frac{x_{ij}^t+\frac{e}{n}}{x_{ij}^{t-1}+\frac{e}{n}} \quad (4) \quad d_t(i,j) + e_{ij}^t \quad a_j^t.$
- $b_{ij}^t = \frac{g}{h} \ln \frac{1+\frac{e}{n}}{x_{ij}^{t-1}+\frac{e}{n}} \quad x^t \quad 1 \quad \frac{g}{h} \ln \frac{1+\frac{e}{n}}{1+\frac{e}{n}} = 0$

The above inequalities prove that the second, third and fourth group of constraints of D also hold, thus completing the proof of the lemma. □

We are now ready to prove Theorem 5.3, by showing that the dual we constructed can pay for the facility, connection and switching cost of the Algorithm 7. Since we bound together the facility cost and connection cost, we will simply refer to them as the service cost. Throughout the proofs, we will use the following relations:

$$a_j (1 \quad \sum_{i=1}^{m} x_{ij}) = 0, \; 8j \; 2 \; [n] \tag{5.5}$$

$$y_i (f \quad \sum_{j=1}^{n} e_{ij}) = 0, \; 8i \; 2 \; [m] \tag{5.6}$$

$$x_{ij}(d_t(i,j) + \frac{g}{h} \ln \frac{x_{ij} + \frac{e}{n}}{x_{ij}^{t-1} + \frac{e}{n}} + e_{ij} \quad a_j) = 0, \; 8i \; 2 \; [m], 8j \; 2 \; [m] \tag{5.7}$$

$$e_{ij}(x_{ij} \quad y_i) = 0, \; 8j \; 2 \; [n], 8i \; 2 \; [m] \tag{5.8}$$

$$h \quad k \quad h \ln(h/k) \text{ for any } h, k > 0 \tag{5.9}$$

$$\sum_i h_i \ln(h_i/k_i) \quad \left(\sum_i h_i \right) \log \frac{\sum_i h_i}{\sum_i k_i} \tag{5.10}$$

Equalities 5.5, 5.6, 5.7, 5.8 are the KKT conditions of P and its dual and the remaining two inequalities are standard logarithmic inequalities. Theorem 5.3 will follow from the next two lemmas. The analysis is similar with that of Theorem 1.1 in [26] adapted to the objective of ODFL and also dealing with the presence of precedence constraints.

Lemma 5.2. *The switching cost M of Algorithm 7 is at most $h(1 + \frac{em}{n})$ times the cost of the dual feasible solution of Lemma 5.1.*

Proof. Let M_t be the switching cost of Algorithm 7 at round t. The summation below is taken over increasing values of connection variables, i.e. $x_{ij}^t > x_{ij}^{t-1}$, since decreasing

values only decrease the fractional switching cost.

$$
\begin{aligned}
M_t &= g \sum \left(x_{ij}^{t} \quad x_{ij}^{t-1} \right) = h \ \frac{g}{h} \sum \left(x_{ij}^{t} \quad x_{ij}^{t-1} \right) \\
&= h \ \frac{g}{h} \sum \left(x_{ij}^{t} + \frac{e}{n} \quad \left(x_{ij}^{t-1} + \frac{e}{n} \right) \right) \\
& \quad h \sum \left(x_{ij}^{t} + \frac{e}{n} \right) \frac{g}{h} \ln \frac{x_{ij}^{t} + \frac{e}{n}}{x_{ij}^{t-1} + \frac{e}{n}} && \text{(by inequality (5.9))} \\
& \quad h \sum_{i=1}^{m} \sum_{j=1}^{n} \left(x_{ij}^{t} + \frac{e}{n} \right) (a_j^{t} \quad e_{ij}^{t} \quad d_t(i,j)) && \text{(by (5.7))} \\
& \quad h \sum_{j=1}^{n} a_j^{t} \left(\sum_{i=1}^{m} \left(x_{ij}^{t} + \frac{e}{n} \right) \right) = h \sum_{j=1}^{n} a_j^{t} \left(\sum_{i=1}^{m} x_{ij}^{t} + \frac{em}{n} \right) \\
&= h \sum_{j=1}^{n} a_j^{t} \left(1 + \frac{em}{n} \right) = h \left(1 + \frac{em}{n} \right) \sum_{j=1}^{n} a_j^{t} && \text{(by (5.5))}
\end{aligned}
$$

Hence,

$$
M = \sum_{t=1}^{T} M_t \quad h \left(1 + \frac{em}{n} \right) \sum_{t=1}^{T} \sum_{j=1}^{n} a_j^{t} \tag{5.11}
$$

This concludes the proof of the lemma. □

Lemma 5.3. *The total service cost* S *of Algorithm 7 is less than the cost of the dual feasible solution of Lemma 5.1.*

Proof.

$$
\begin{aligned}
S &= \sum_{t=1}^{T}\left[f\sum_{i=1}^{m} y_i^{\,t} + \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}^{\,t} d_t(i,j)\right] \\
&= \sum_{t=1}^{T}\left[\sum_{i=1}^{m} y_i^{\,t}\left(\sum_{j=1}^{n} e_{ij}^{\,t}\right) + \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}^{\,t}\left(a_j^{\,t} \quad e_{ij}^{\,t} \quad \frac{g}{h}\ln\frac{x_{ij}^{\,t}+\frac{e}{n}}{x_{ij}^{\,t\ 1}+\frac{e}{n}}\right)\right] \\
&\qquad\qquad \text{(by (5.7) and (5.6))} \\
&= \sum_{t=1}^{T}\left[\sum_{i=1}^{m}\sum_{j=1}^{n}(y_i^{\,t} \quad x_{ij}^{\,t})e_{ij}^{\,t} + \sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}^{\,t} a_j^{\,t} \quad \frac{g}{h}\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}^{\,t}\ln\frac{x_{ij}^{\,t}+\frac{e}{n}}{x_{ij}^{\,t\ 1}+\frac{e}{n}}\right] \\
&= \sum_{t=1}^{T}\left[\sum_{j=1}^{n} a_j^{\,t}\left(\sum_{i=1}^{m} x_{ij}^{\,t}\right) \quad \frac{g}{h}\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}^{\,t}\ln\frac{x_{ij}^{\,t}+\frac{e}{n}}{x_{ij}^{\,t\ 1}+\frac{e}{n}}\right] \qquad \text{(by (5.8))} \\
&= \sum_{t=1}^{T}\sum_{j=1}^{n} a_j^{\,t} \quad \frac{g}{h}\sum_{j=1}^{n}\sum_{i=1}^{m}\left[\sum_{t=1}^{T}\left(x_{ij}^{\,t}+\frac{e}{n}\right)\ln\frac{x_{ij}^{\,t}+\frac{e}{n}}{x_{ij}^{\,t\ 1}+\frac{e}{n}} \quad \frac{e}{n}\sum_{t=1}^{T}\ln\frac{x_{ij}^{\,t}+\frac{e}{n}}{x_{ij}^{\,t\ 1}+\frac{e}{n}}\right] \\
&\qquad\qquad \text{(by (5.5))} \\
&\qquad \sum_{t=1}^{T}\sum_{j=1}^{n} a_j^{\,t} \quad \frac{g}{h}\sum_{j=1}^{n}\sum_{i=1}^{m}\left[\sum_{t=1}^{T}\left(x_{ij}^{\,t}+\frac{e}{n}\right)\ln\frac{\sum_{t=1}^{T}(x_{ij}^{\,t}+\frac{e}{n})}{\sum_{t=1}^{T}(x_{ij}^{\,t\ 1}+\frac{e}{n})} \quad \frac{e}{n}\ln\frac{x_{ij}^{\,T}+\frac{e}{n}}{x_{ij}^{\,0}+\frac{e}{n}}\right] \\
&\qquad\qquad \text{(by (5.10))}
\end{aligned}
$$

Notice that that the two terms in the bracket of the right hand side of the inequality above cancel each other out, since:

$$
\frac{e}{n}\ln\frac{x_{ij}^{\,T}+\frac{e}{n}}{x_{ij}^{\,0}+\frac{e}{n}} \overset{x_{ij}^{\,0}=0}{=} \left(x_{ij}^{\,0}+\frac{e}{n}\right)\ln\frac{x_{ij}^{\,0}+\frac{e}{n}}{x_{ij}^{\,T}+\frac{e}{n}} \overset{(5.9)}{} x_{ij}^{\,0} \quad x_{ij}^{\,T}
$$

$$
\left(\sum_{t=1}^{T}(x_{ij}^{\,t}+\frac{e}{n})\right)\ln\frac{\sum_{t=1}^{T}(x_{ij}^{\,t}+\frac{e}{n})}{\sum_{t=1}^{T}(x_{ij}^{\,t\ 1}+\frac{e}{n})} \overset{(5.9)}{} \sum_{t=1}^{T}\left(x_{ij}^{\,t}+\frac{e}{n}\right) \quad \sum_{t=1}^{T}\left(x_{ij}^{\,t\ 1}+\frac{e}{n}\right) = x_{ij}^{\,0} \quad x_{ij}^{\,T}
$$

Therefore, it holds that

$$
S \quad \sum_{t=1}^{T}\sum_{j=1}^{n} a_j^{\,t}
$$

□

We can now easily prove the performance of Algorithm 7 stated in Theorem 5.3.

Algorithm 8 The rounding algorithm

1: **Initialization:** Choose i.i.d. random variables $Z_{ij} \quad \exp(1), \forall i, j$.
2: **At each round t:**
3: Let x_{ij}^t be the fractional value of round t obtained by Algorithm 7 .
4: For each client j, open $i = \arg\min_{i'} \frac{Z_{i'j}}{x_{i'j}^t}$ and connect j to i.

Proof of Theorem 5.3. Let OPT(D) and OPT(P) denote the optimal solutions of the D and P respectively. By Lemma 5.2 and Lemma 5.3, the total cost of Algorithm 7 is:

$$
\begin{aligned}
S + M &\quad \left[1 + h\left(1 + \frac{em}{n}\right)\right] \sum_{t=1}^{T} \sum_{j=1}^{n} a_j{}^t \\
&= \left[1 + \ln\left(1 + \frac{n}{e}\right)\left(1 + \frac{em}{n}\right)\right] \sum_{t=1}^{T} \sum_{j=1}^{n} a_j{}^t \\
&\quad \left[1 + \ln\left(1 + \frac{n}{e}\right)\left(1 + \frac{em}{n}\right)\right] \mathrm{OPT(D)} && \text{(by Lemma 5.1)} \\
&= \left[1 + (1 + e^0) \ln\left(1 + \frac{m}{e^0}\right)\right] \mathrm{OPT(D)} && \text{(since } e^0 = \tfrac{em}{n}\text{)} \\
&= \left[1 + (1 + e^0) \ln\left(1 + \frac{m}{e^0}\right)\right] \mathrm{OPT(P)}
\end{aligned}
$$

□

The proof of Theorem 5.3 concludes this section.

5.4 The Rounding Algorithm

In this section, we present Algorithm 8, which makes use of the exponential distribution to round the fractional solution to an integral solution at each round. The analysis shows that the fractional solution grows up to a factor logarithmic in n regarding the facility cost and up to constant factors regarding the switching and connection cost. Before proceeding to the details of Algorithm 8, we give the definition of an exponential random variable and some of their properties.

Definition 5.4. *A random variable X is distributed according to the exponential distribution with rate l, denoted as $X \quad \exp(l)$, if it has density $f_X(x) = l e^{lx}$ for every $x \quad 0$, and $f_X(x) = 0$ otherwise. We will use the following properties of exponential random variables:*

1. *If $X \quad \exp(l)$ and $c > 0$, then $X/c \quad \exp(lc)$.*
2. *Let $X_1, \ldots, X_k$ be independent random variables with $X_i \quad \exp(l_i)$:*
   (a) $\min\{X_1, \ldots, X_k\} \quad \exp(l_1 + \ldots + l_k)$
   (b) $\Pr[X_i \quad \min_{j \neq i} X_j] = \frac{l_i}{l_1 + \ldots + l_k}$

3. *If* $X \quad \exp(l)$ *and* $Y \quad \exp(m)$ *are independent, then* $8t \quad 0$*:*

$$\Pr[X \quad Y \; j \; X \quad t] = \frac{l}{l+m} \quad e^{mt}$$

The rounding algorithm samples independently a total of $n \quad m$ (one for each client-facility connection) random variables Z_{ij} from the exponential distribution with rate $l = 1$ at the beginning of its execution, which will be used throughout all rounds. Then, at each round t, it chooses for each client j the connection fi, jg minimizing the ratio $\frac{Z_{ij}}{x_{ij}^t}$, where x_{ij}^t is the fractional variable of this connection obtained by Algorithm 7. Notice that by the properties of Definition 5.4 the ratio $\frac{Z_{ij}}{x_{ij}^t}$ is also an exponential random variable. This technique is referred to as competing exponential clocks, since a random variable wins the competition if it has the smallest value among all others (minimizes the ratio $\frac{Z_{ij}}{x_{ij}^t}$ in our case).

The high level idea of the analysis is that connection and switching cost of the rounded solution add only constant factors to the cost of the connection and switching cost of the fractional solution at each round t. The reason is that they favor connections to facilities that are dependent to the increase/decrease of the fractional variables x_{ij}^t. This fact combined with the properties of the exponential distribution leads to a rounding of the *right* connections indicated by the fractional solution. On the other side, this leads to more open facilities, since we prove that the rounding adds a factor logarithmic in n to the cost of the fractional solution.

Next, we will analyze the performance of Algorithm 8 by bounding separately the facility, connection, and switching cost. We will simply calculate the probabilities of opening any facility, connecting a client to a facility and changing a connection.

Facility cost

We start with the facility cost of the rounding algorithm, which is $O(\log n)$-competitive with respect to the facility cost of the fractional solution.

Proof. Let E_{ij} denote the event that $i = \arg\min_{i'} \frac{Z_{i'j}}{x_{i'j}}$ for some client j and let $a > 0$ be chosen later. The probability of E_{ij} equals:

$$
\begin{aligned}
\Pr[\exists j : E_{ij}] &= \Pr\left[\exists j : E_{ij} \wedge \frac{Z_{ij}}{x_{ij}} < a\right] \Pr\left[\frac{Z_{ij}}{x_{ij}} < a\right] \\
&\quad + \Pr\left[\exists j : E_{ij} \wedge \frac{Z_{ij}}{x_{ij}} \geq a\right] \Pr\left[\frac{Z_{ij}}{x_{ij}} \geq a\right] \\
&\quad \Pr\left[\frac{Z_{ij}}{x_{ij}} < a\right] + \Pr\left[\exists j : E_{ij} \wedge \frac{Z_{ij}}{x_{ij}} \geq a\right] \Pr\left[\frac{Z_{ij}}{x_{ij}} \geq a\right] \\
&\quad \Pr\left[\frac{Z_{ij}}{x_{ij}} < a\right] \\
&\quad + \sum_{j=1}^{n} \Pr\left[E_{ij} \wedge \frac{Z_{ij}}{x_{ij}} \geq a\right] \Pr\left[\frac{Z_{ij}}{x_{ij}} \geq a\right] \qquad \text{(By the union bound)} \\
&= 1 \quad e^{-ax_{ij}} + \sum_{j=1}^{n} \Pr\left[E_{ij} \wedge \frac{Z_{ij}}{x_{ij}} \geq a\right] e^{-ax_{ij}} \\
&\quad ax_{ij} + \sum_{j=1}^{n} \frac{x_{ij}}{\sum_{i=1}^{m} x_{ij}} e^{a\left(\sum_{j' \neq j} x_{ij'} - x_{ij}\right)} e^{-ax_{ij}} \\
&\qquad \text{(} 1 - e^{-x} \leq x, \forall x \text{ and by Definition 5.4)} \\
&\quad ax_{ij} + \sum_{j=1}^{n} e^{-a} x_{ij} \leq ay_i + ne^{-a} y_i. \qquad \text{(since } x_{ij} \leq y_i\text{)}
\end{aligned}
$$

By choosing $a = \log n$ we have the result. □

Connection cost

Next, we show that the connection cost of the rounding algorithm is $O(1)$-competitive with the connection cost of the fractional solution. Again, let $a > 0$ be chosen later.

Proof. Similar arguments with the previous proof, show that the probability to choose connection ij is:

$$\Pr[ij] \quad \Pr\left[\frac{Z_{ij}}{x_{ij}} < a\right] + \Pr\left[\frac{Z_{ij}}{x_{ij}} = \min_{i'} \frac{Z_{i'j}}{x_{i'j}} \; j \; \frac{Z_{ij}}{x_{ij}} \quad a\right] \quad \Pr\left[\frac{Z_{ij}}{x_{ij}} \quad a\right]$$

$$1 \quad e^{\;ax_{ij}} + \Pr\left[\frac{Z_{ij}}{x_{ij}} = \min_{i'} \frac{Z_{i'j}}{x_{i'j}} \; j \; \frac{Z_{ij}}{x_{ij}} \quad a\right] e^{\;ax_{ij}}$$

$$(1 \quad e^{\;x} \quad x, \forall x \text{ and by Definition 5.4})$$

$$ax_{ij} + \frac{x_{ij}}{\sum_{i=1}^{m} x_{ij}} e^{\;a\left(\sum_{i' \neq i} x_{i'j} \quad x_{ij}\right)} e^{\;ax_{ij}}$$

$$ax_{ij} + e^{\;a} x_{ij}.$$

By choosing a sufficiently small a (for example $a = 1$) , we have the result. □

Switching cost

Finally, we show that every step that incurs a fractional switching cost of d in a connection variable x_{ij}, incurs an expected increase of at most $\frac{d}{d+1}$ in the randomized solution. Thus, the expected number of new connections is $O(1)$.

Proof. We break down the total movement from time $t \quad 1$ to t in the fractional solution into $m \quad n$ intermediate steps, on each of which only the value of exactly one x_{ij} is changed. We take first all the x_{ij}'s whose value increases and then all the x_{ij}'s whose value decreases, thus managing to preserve a feasible solution in all the intermediate steps. This way, the total switching cost from time $t \quad 1$ to time t of the fractional solution does not change while the integral switching cost could only increase due to possible changes in the intermediate steps. First, we will prove the bound in the case the connection variable decreases: $x_{ij}^t = x_{ij}^{t \;\; 1} \quad d$. Let $Y_{ij} = \min_{i' \neq i} \frac{Z_{i'j}}{x_{i'j}^t}$, where $Y_{ij} \quad \exp(l)$ for $l = 1 \quad x_{ij}^t$. When the value of x_{ij} decreases, the value of $\frac{Z_{ij}}{x_{ij}}$ increases. Therefore, connection ij cannot be chosen a time t if is not chosen at time $t \quad 1$. However, due to the increase of $\frac{Z_{ij}}{x_{ij}}$, another connection could turn minimal that had not been chosen in the previous time step. This is the only case, when a switching cost is incurred. The probability of this event is bounded by:

$$\Pr\left[\frac{Z_{ij}}{x_{ij}^{t \;\; 1}} \quad Y_{ij} \quad \frac{Z_{ij}}{x_{ij}^{t \;\; 1} \quad d}\right] = F_{Y_{ij}}[¥] \quad F_{Y_{ij}}[0] = \frac{l}{x_{ij}^{t \;\; 1} \quad d + l} \quad \frac{l}{x_{ij}^{t \;\; 1} + l}$$

This expression is maximized when $x_{ij}^{t \;\; 1} \quad d = 0, l = 1$, therefore is less than

$$1 \quad \frac{1}{d+1} = \frac{d}{d+1}$$

Now, we turn to the case where $x_{ij}^t = x_{ij}^{t-1} + d$. When x_{ij} increases, the ratio $\frac{z_{ij}}{x_{ij}}$ decreases. Therefore, if facility i was chosen in the previous step it will be chosen again in this step, thus not incurring switching cost. If facility was not chosen in the previous step, it will be chosen in this step with probability

$$\Pr\left[\frac{z_{ij}}{x_{ij}^{t-1}+d} \quad Y_{ij} \quad \frac{z_{ij}}{x_{ij}^{t-1}}\right]$$

which is no more than $\frac{d}{d+1}$, following the exact same analysis with the case of the decreasing connection variables. □

Finally, it is easy to provide the proof of Theorem 5.2, which concludes this section.

Proof of Theorem 5.2. First notice that by Lemma 5.3 the fractional solution of Algorithm 7 is optimal with respect to the facility and connection cost. Therefore, Algorithm 8 will round the solution to an integral one only losing a factor of $O(\log n)$ in the facility cost and a factor $O(1)$ in the connection cost, thus being $O(\log n)$ competitive with the optimal offline solution. Regarding the switching cost, by Lemma 5.2 the fractional solution is $O(\log m)$ competitive with the fractional solution. The cost of this solution will only increase by a factor of $O(1)$ after the randomized rounding, thus proving the result. □

Bibliography

[1] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998.

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

[3] C. Ambühl. *On the list update problem.* PhD thesis, ETH Zurich, 2002.

[4] H. An, A. Norouzi-Fard, and O. Svensson. Dynamic facility location via exponential clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20, 2017.

[5] H.-C. An, A. Norouzi-Fard, and O. Svensson. Dynamic facility location via exponential clocks. *ACM Transactions on Algorithms (TALG)*, 13(2):21, 2017.

[6] A. Anagnostopoulos, R. Bent, E. Upfal, and P. V. Hentenryck. A simple and deterministic competitive algorithm for online facility location. *Inf. Comput.*, 194(2):175–202, 2004.

[7] E. Anshelevich and J. Postl. Randomized social choice functions under metric preferences. *J. Artif. Intell. Res.*, 58:797–827, 2017.

[8] C. J. Argue, A. Gupta, G. Guruganesh, and Z. Tang. Chasing convex bodies with linear competitive ratio. In *SODA*, pages 1519–1524, 2020.

[9] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[10] Y. Azar and I. Gamzu. Ranking with submodular valuations. In *SODA*, pages 1070–1079, 2011.

[11] Y. Azar, I. Gamzu, and X. Yin. Multiple intents re-ranking. In *STOC*, pages 669–678, 2009.

[12] N. Bansal, N. Buchbinder, and J. Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012.

[13] N. Bansal, M. Eliáš, Ł. Jeż, G. Koumoutsos, and K. Pruhs. Tight bounds for double coverage against weak adversaries. volume 62, pages 349–365. Springer, 2018.

[14] N. Bansal, M. Eliáš, and G. Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *FOCS*, pages 493–504, 2017.

[15] N. Bansal, M. Eliáš, G. Koumoutsos, and J. Nederlof. Competitive algorithms for generalized *k*-server in uniform metrics. In *SODA*, pages 992–1001, 2018.

[16] N. Bansal, M. Eliéš, Ł. Jeż, and G. Koumoutsos. The (h, k)-server problem on bounded depth trees. volume 15, page 28. ACM, 2019.

[17] N. Bansal, A. Gupta, and R. Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA*, pages 1539–1545, 2010.

[18] A. Bar-Noy, M. Bellare, M. M. Halldórsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.

[19] A. Bar-Noy, M. M. Halldórsson, and G. Kortsarz. A matched approximation bound for the sum of a greedy coloring. *Inf. Process. Lett.*, 71(3-4):135–140, 1999.

[20] Y. Bartal and E. Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theoretical Computer Science*, 324(2):337 – 345, 2004.

[21] Y. Bartal and E. Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. volume 324, pages 337–345. Elsevier, 2004.

[22] A. Blum and C. Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.

[23] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[24] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[25] S. Bubeck, M. B. Cohen, J. R. Lee, and Y. T. Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *SODA*, pages 89–97. SIAM, 2019.

[26] N. Buchbinder, S. Chen, and J. Naor. Competitive analysis via regularization. In C. Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 436–444. SIAM, 2014.

[27] N. Buchbinder, S. Chen, J. Naor, and O. Shamir. Unified algorithms for online learning and competitive analysis. In S. Mannor, N. Srebro, and R. C. Williamson, editors, *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, volume 23 of *JMLR Proceedings*, pages 5.1–5.18. JMLR.org, 2012.

[28] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

[29] M. Charikar and R. Panigrahy. Clustering to minimize the sum of cluster diameters. *Journal of Computer and System Sciences*, 68(2):417–441, 2004.

[30] M. Chrobak, H. J. Karloff, T. H. Payne, and S. Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

[31] M. Chrobak and L. L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991.

[32] C. Coester and E. Koutsoupias. The online k-taxi problem. *To appear in STOC 2019*, 2019.

[33] C. Coester, E. Koutsoupias, and P. Lazos. The infinite server problem. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[34] C. Coester and J. R. Lee. Pure entropic regularization for metrical task systems. In *COLT*, volume 99 of *Proceedings of Machine Learning Research*, pages 835–848. PMLR, 2019.

[35] B. de Keijzer and D. Wojtczak. Facility reallocation on the line. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 188–194, 2018.

[36] G. Divéki and C. Imreh. Online facility location with facility movements. *Central European Journal of Operations Research*, 19(2):191–200, 2011.

[37] D. Eisenstat, C. Mathieu, and N. Schabanel. Facility location in evolving metrics. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.

[38] R. El-Yaniv. *There are infinitely many competitive-optimal online list accessing algorithms*. Hebrew University of Jerusalem, 1996.

[39] U. Feige, L. Lovász, and P. Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[40] B. Feldkord and F. Meyer auf der Heide. Online facility location with mobile facilities. In C. Scheideler and J. T. Fineman, editors, *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 373–381. ACM, 2018.

[41] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms (extended abstract). In *31st Annual Symposium on Foundations of Computer Science*, pages 454–463, 1990.

[42] D. Fotakis. Incremental algorithms for facility location and k-median. *Theoretical Computer Science*, 361(2-3):275–313, 2006.

[43] D. Fotakis. On the Competitive Ratio for Online Facility Location. *Algorithmica*, 50(1):1–57, 2008.

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

[45] D. Fotakis, L. Kavouras, P. Kostopanagiotis, P. Lazos, S. Skoulakis, and N. Zarifis. Reallocating multiple facilities on the line. In S. Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 273–279. ijcai.org, 2019.

[46] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997.

[47] Z. Friggstad and M. R. Salavatipour. Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms (TALG)*, 7(3):28, 2011.

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

[49] R. Hassin and A. Levin. An approximation algorithm for the minimum latency set cover problem. In *ESA*, pages 726–733, 2005.

[50] M. Henzinger, D. Leniowski, and C. Mathieu. Dynamic Clustering to Minimize the Sum of Radii. In K. Pruhs and C. Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:10, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[51] D. S. Hochbaum. Heuristics for the fixed cost median problem. *Math. Program.*, 22(1):148–162, 1982.

[52] S. Im, V. Nagarajan, and R. van der Zwaan. Minimum latency submodular cover. *ACM Trans. Algorithms*, 13(1):13:1–13:28, 2016.

[53] S. Im, M. Sviridenko, and R. van der Zwaan. Preemptive and non-preemptive generalized min sum set cover. *Math. Program.*, 145(1-2):377–401, 2014.

[54] E. Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

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

[56] S. Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013.

[57] E. Liberty, R. Sriharsha, and M. Sviridenko. An algorithm for online k-means clustering. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 81–89. SIAM, 2016.

[58] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, 1994.

[59] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 322–333. ACM, 1988.

[60] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990.

[61] R. Meir, A. D. Procaccia, J. S. Rosenschein, and A. Zohar. Complexity of strategic behavior in multi-winner elections. *J. Artif. Intell. Res.*, 33:149–178, 2008.

[62] A. Meyerson. Online facility location. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 426. IEEE Computer Society, 2001.

[63] M. Sellke. Chasing convex bodies optimally. In *SODA*, pages 1509–1518, 2020.

[64] R. Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM J. Comput.*, 43(1):96–125, 2014.

[65] M. Skutella and D. P. Williamson. A note on the generalized min-sum set cover problem. *Oper. Res. Lett.*, 39(6):433–436, 2011.

[66] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.