Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογιστών

Αλγόριθμοι Προσέγγισης για το πρόβλημα Δυναμικής Κάλυψης Συνόλου Ελάχιστου Αθροίσματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΑΝΑΓΙΩΤΗΣ ΚΩΣΤΟΠΑΝΑΓΙΩΤΗΣ

Επιβλέπων : Δημήτριος Φωτάκης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Λογικής και Επιστήμης Υπολογιστών

Αλγόριθμοι Προσέγγισης για το πρόβλημα Δυναμικής Κάλυψης Συνόλου Ελάχιστου Αθροίσματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΑΝΑΓΙΩΤΗΣ ΚΩΣΤΟΠΑΝΑΓΙΩΤΗΣ

Επιβλέπων : Δημήτριος Φωτάκης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16η Ιουλίου 2021.

....	....	....
Δημήτριος Φωτάκης Αναπληρωτής Καθηγητής Ε.Μ.Π.	Αριστείδης Παγουρτζής Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Καθηγητής Ε.Μ.Π

Αθήνα, Ιούλιος 2021

....

Παναγιώτης Κωστοπαναγιώτης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στην παρούσα διπλωματική εργασία εξετάζουμε το πρόβλημα δυναμικής κάλυψης συνόλου ελάχιστου αθροίσματος (Mult-MSSC) μια φυσιολογική και ενδιαφέρουσα γενίκευση του κλασικού προβλήματος ανανέωσης λίστας. Στο πρόβλημα Mult-MSSC πρέπει να διατηρήσουμε μια ακολουθία μεταθέσεων $(\pi^0, \pi^1, \ldots, \pi^T)$ n τω πλήθος στοιχείων βάσει μιας ακολουθίας συνόλων κάλυψης $\mathcal{R} = (R^1, \ldots, R^T)$. Στόχος μας είναι να ελαχιστοποιήσουμε το κόστος ανανέωσης απ' την μετάθεση π^{t-1} στην π^t, το οποίο ποσοτικοποιείται με την απόσταση Kendall Tau $\mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^t)$, συν το συνολικό κόστος κάλυψης κάθε αιτήματος R^t με την παρούσα μετάθεση π^t που είναι στην ουσία η θέση του πρώτου στοιχείου του συνόλου R^t στην μετάθεση π^t.

Ξεκινάμε με μια ιστορική αναδρομή του προβλήματος παρουσιάζοντας τις διάφορες εκδοχές του προβλήματος που έχουν παρουσιαστεί στην βιβλιογραφία και τους αλγορίθμος που χρησιμοποιούνται για τον σχεδιασμό αλγορίθμων για την κάθε εκδοχή. Η παρουσίαση αυτών των τεχνικών είναι πολύ σημαντική καθώς κτίζουν μια καλή διαίσθηση για το πρόβλημα και επιπλέον βοηθάνε στην κατανόηση των αποτελεσμάτων της offline Dynamic εκδοχής με την οποία και ασχολούμαστε.

Σε επόμενη φάση, συνεχίζουμε με μια ιστορική αναδρομή καθώς και παραλλαγές του προβλήματος. Ξεκινάμε με το πρόβλημα ανανέωσης λίστας(List-Update) και εξηγούμε πως χρησιμοποιώντας την μέθοδο δυναμικού που περιγράψαμε στην προηγούμενη ενότητα μπορούμε να λύσουμε το πρόβλημα με παράγοντα προσέγγισης 2. Επιπλέον, παρουσιάζουμε μια γενικότερη οικογένεια προβλημάτων που εξελίσσονται στον χρόνο και εξηγούμε πως μπορούμε να χρησιμοποιήσουμε Rounding γραμμικών προγραμμάτων προκειμένου να λάβουμε ακριβείς ή και προσεγγιστικές λύσεις για ορισμένα προβλήματα. Χρησιμοποιούμε το πρόβλημα Facility Reallocation on the Real Line για την παρουσίαση αυτής της τεχνικής.

Στην τελευταία ενότητα παρουσιάζουμε τα αποτελέσματά μας για το πρόβλημα Mult-MSSC. Ανάγοντας απ' το κλασικό πρόβλημα Κάλυψης Συνόλου(Set Cover), δείχνουμε αρχικά ότι το Mult-MSSC δεν μπορεί να προσεγγιστεί με παράγοντα προσέγγισης $O(1)$ εκτός και αν $\mathrm{P} = \mathrm{NP}$ καθώς και ότι οποιοσδήποτε αλγόριθμος με παράγοντα προσέγγισης $o(\log n)$ (αντίστοιχα $O(r)$) για το Mult-MSSC θα έδινε υπολογαριθμικό παράγοντα προσέγγισης για το Set Cover (ή (r) αντίστοιχα αν θεωρήσουμε την εκδοχή όπου κάθε στοιχείο εμφανίζεται το πολύ r φορές στα σύνολα κάλυψης). Η βασική μας συνεισφορά έγκειται στο οτι δείχνουμε πως το πρόβλημα Mult-MSSC μπορεί να προσεγγιστεί σε πολυωνυμικό χρόνο με παράγοντα $O(\log^2 n)$ στην γενική περίπτωση με randomized rounding και με παράγοντα $O(r^2)$, αν όλα τα σύνολα αιτημάτων κάλυψης έχουν πληθικότητα το πολύ r, με deterministic rounding.

Λέξεις κλειδιά

Κάλυμα συνόλου, Δυναμικοί Αλγόριθμοι, Αλγόριθμοι Προσέγγισης, Online Αλγόριθμοι, Combinatorial Optimization

Abstract

We investigate the polynomial-time approximability of the multistage version of Min-Sum Set Cover (Mult-MSSC), a natural and intriguing generalization of the classical List Update problem. In Mult-MSSC, we maintain a sequence of permutations $(\pi^0, \pi^1, \ldots, \pi^T)$ on n elements, based on a sequence of requests $\mathcal{R} = (R^1, \ldots, R^T)$. We aim to minimize the total cost of updating π^{t-1} to π^t, quantified by the Kendall tau distance $\mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^t)$, plus the total cost of covering each request R^t with the current permutation π^t, quantified by the position of the first element of R^t in π^t.

We begin by examining the history of the problem and the main versions that have appeared in literature over time. Presenting the main techniques use for the design of algorithms is crucial since they build a nice intuition regarding the problem and pave the way to understand the results of the offline Dynamic version which is essentially the core of this thesis.

In the next chapter we continue with the history of the problem. By diving into the List Update problem we explain how the use of the Potential Method can be used to solve the problem with a 2-approximation algorithm. Furthermore, we present a general family of problems that are time evolving and show how we can use Rounding of appropriate Linear Programs to obtain exact or approximation algorithms for these problems. To demonstrate this technique we use the Facility Reallocation on the Real Line problem.

Using a reduction from Set Cover, we show that Mult-MSSC does not admit an $O(1)$-approximation, unless P $=$ NP, and that any $o(\log n)$ (resp. $o(r)$) approximation to Mult-MSSC implies a sublogarithmic (resp. $o(r)$) approximation to Set Cover (resp. where each element appears at most r times). Our main technical contribution is to show that Mult-MSSC can be approximated in polynomial-time within a factor of $O(\log^2 n)$ in general instances, by randomized rounding, and within a factor of $O(r^2)$, if all requests have cardinality at most r, by deterministic rounding.

Key words

Set Cover, Dynamic Algorithms, Approximation Algorithms, Online Algorithms, Combinatorial Optimization

Ευχαριστίες

Παναγιώτης Κωστοπαναγιώτης,

Αθήνα, 16η Ιουλίου 2021

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-42-17, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιούλιος 2021.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Κεφάλαιο 1

Εκτεταμένη Περίληψη στα Ελληνικά

Στην παρούσα διπλωματική εργασία μελετάμε το πρόβλημα Multistage Min-Sum Set Cover, μια εκδοχή του γνωστού και μελετημένου προβλήματος Min-Sum Set Cover που μέχρι στιγμής δεν έχει εμφανιστεί στην βιβλιογραφία. Πρωτού περάσουμε στην παρουσιάση των αποτελεσμάτων μας είναι σημαντικό να κάνουμε μια αναδρομή στην ιστορία του προβλήματος και να δείξουμε τις βασικές τεχνικές που χρησιμοποιούνται στον σχεδιασμό προσεγγιστικών αλγορίθμων για κάθε διαφορετική παραλλαγή.

1.1 Η Στατική Εκδοχή του Min-Sum Set Cover

Στην παρούσα εκδοχή του προβλήματος μας δίνεται μια συλλογή από n σύνολα που συνολικά καλύπτουν m στοιχεία. Θέλουμε να βρούμε μια μετάθεση των m στοιχείων ώστε να ελαχιστοποιήσουμε το συνολικό κόστος κάλυψης των συνόλων. Το κόστος κάλυψης για κάθε σύνολο είναι η θέση του πρώτου στοιχείου του συνόλου στην μετάθεση που έχουμε επιλέξει.

Συγκεκριμένα, ορίζουμε το πρόβλημα Min Sum Set Cover ως εξής. Έστω το σύνολο στοιχείων U με $|U| = n$, χωρίς βλάβη της γενικότητας θεωρούμε $U = \{1, 2, ..., n\} = [1, n]$. Επιπλέον, έστω m σύνολα $S_1, S_2, ..., S_m \subseteq U$. Ο σκοπός μας είναι να βρούμε μια μετάθεση των n στοιχείων που ελαχιστοποιούν το συνολικό κόστος των συνόλων S_i, δηλαδή το $\sum_{i=1}^{m} C(S_i)$ όπου το κόστος $C(S_i)$ κάθε συνόλου είναι η θέση του στοιχείου του συνόλου που εμφανίζεται πρώτο στην μετάθεση.

Το πρόβλημα Min Sum Set Cover έχει μελετηθεί εκτεταμμένα. Παρακάτω παρουσιάζουμε τον βασικό αλγόριθμο και τα αποτελέσματα όπως αυτά υπάρχουν στο [24].

1.1.1 Ο Άπληστος Αλγόριθμος

Ο άπληστος αλγόριθμος για το Min Sum Set Cover είναι πολύ φυσιολογικός και απλός.

Algorithm 1 Άπληστος Αλγόριθμος για το Min Sum Set Cover

```
Είσοδος: Ένα σύνολο $U$ καθώς και μια συλλογή m συνόλων $S_1, S_2, ..., S_m$
Έξοδος: Μια μετάθεση των στοιχείων του $U$.
1: $\pi = [\,]$
2: while $\pi$ δεν περιέχει όλα τα στοιχεία του $U$ do
3:     Βάλε στο τέλος του $\pi$ το στοιχείο του U που εμφανίζεται στα περισσότερα σύνολα.
4: end while
5: return $\pi$
```

1.1.2 Βασικά Αποτελέσματα

Παρόλο που ο αλγόριθμος είναι πολύ απλός στο [24] οι συγγραφείς δείχνουν τα ακόλουθα ισχυρά αποτελέσματα.

Θεώρημα 1. *Ο Αλγόριθμος 5 προσεγγίζει το min sum set cover με παράγοντα προσέγγισης το πολύ 4.*

Μιας και αυτός ο αλγόριθμος είναι απλός και φυσιολογικός ένα ερώτημα που δημιουργείται είναι το κατά πόσο κάποια πιο περίπλοκη τεχνική θα μπορούσε να δώσει αλγόριθμο με καλύτερο παράγοντα προσέγγισης από 4. Σε αυτό έρχεται να απαντήσει το ακόλουθο θεώρημα.

Θεώρημα 2. *Για κάθε $\epsilon > 0$ είναι NP-Hard να προσεγγίσεις το min sum set cover με παράγοντα προσέγγισης $4 - \epsilon$.*

1.2 Η Γενικευμένη Εκδοχή του Min Sum Set Cover

Μετά την παρουσιάση της στατικής εκδοχής του Min Sum Set Cover συνεχίζουμε με την γενίκευση του προβλήματος που ορίστηκε πρώτη φορά απ' τους Azar, Gamzu και Yin στο [7]. Σε αυτό το πρόβλημα μας δίνεται πάλι ένα σύνολο U με $|U| = n$ και μια σςυλλογή συνόλων $S_1, S_2, ..., S_m$ καθώς και έναν αριθμό για κάθε σύνολο $K(S_i)$. Σκοπός μας είναι να υπολογίσουμε μια μετάθεση του συνόλου U ώστε να ελαχιστοποιήσουμε το συνολικό κόστος κάλυψης όπως και στην απλή εκδοχή του προβλήματος $\sum_{i=1}^{m} C(S_i)$ όπου το κόστος κάλυψης $C(S_i)$ είναι η θέση του $K(S_i)$-οστού στοιχείου στην μετάθεση.

Στο [7] οι συγγραφείς παρουσιάζουν έναν Αλγόριθμο με παράγοντα προσέγγισης $O(logr)$ όπου $r = max_i\{|S_i|\}$. Έπειτα, δόθηκε ένας αλγόριθμος με σταθερό παράγοντα προσέγγισης [10] βασισμένος σε γραμμικό προγραμματισμό.

Σε αυτή την ενότητα παρουσιάζουμε τα αποτελέσματα του [39] όπου με την χρήση α-point scheduling και κατάλληλη χρήση γραμμικού προγραμματισμού βελτιώνεται σημαντικά ο παράγοντας προσέγγισης του [10]. Η παρουσίαση των εν λόγω αποτελεσμάτων είναι σημαντική καθώς οι τεχνικές που χρησιμοποιήθηκαν στο [39] χρησιμοποιούνται παραλλαγμένες για την εξαγωγή των δικών μας αποτελεσμάτων για το Multistage Min-Sum Set Cover.

Το πρώτο βήμα στον σχεδιασμό του αλγορίθμου για το πρόβλημα είναι η επίλυση του ακόλουθου Linear Programming Relaxation. Ορίζουμε τις μεταβλητές $y_{i,t}$ η οποία θέλουμε να είναι 1 αν και μόνο αν $C(S_i) < t$ (δηλαδή το $K(S_i)$-οστό στοιχείο του συνόλου S_i εμφανίζεται πριν την θέση t) και επιπλέον τις μεταβλητές $x_{e,t}$ που θέλουμε να είναι 1 αν και μόνο αν το στοιχείο $e \in U$ βρίσκεται στην θέση t. Βάσει των παραπάνω προκύπτει το ακόλουθο γραμμικό πρόγραμμα.

$$\begin{aligned}
\min \quad & \sum_{t=1}^{n}\sum_{i=1}^{m}(1 - y_{i,t}) \\
\text{s.t} \quad & \sum_{e \in U} x_{e,t} = 1 \quad \text{για κάθε } t \le n \\
& \sum_{t=1}^{n} x_{e,t} = 1 \quad \text{για κάθε } e \in U \\
& \sum_{e \in S \setminus A}\sum_{t' < t} x_{e,t} \ge (K(S_i) - |A|) \cdot y_{i,t} \quad \text{για κάθε } i \le m, A \subseteq S_i, t \le n \\
& x_{e,t}, y_{i,t} \in [0,1] \quad \text{για κάθε } e \in U, i \le m, t \le n
\end{aligned}$$

Το γραμμικό πρόγραμμα φαίνεται εκ πρώτης όψης να έχει εκθετικά πολλά constraints, αφού δημιουργούμε ένα constraint για κάθε υποσύνολο του κάθε συνόλου S_i. Όπως μπορούμε να δούμε και στο [10] τα εκθετικά πολλα constraints μπορούν σε πολυωνυμικό χρόνο να διαχωριστούν ώστε να μπορούμε να υπολογίσουμε αποδοτικά την βέλτιστη λύση του γραμμικού προγράμματος.

Ο βασικός αλγόριθμος για την επίλυση του προβλήματος με παράγοντα προσέγγισης 28 βασίζεται στην μέθοδο α point scheduling. Συγκεκριμένα, για κάθε στοιχείο $e \in U$ επιλέγουμε ανεξάρτητα και ομοιόμορφα $\alpha_e \in [0, 1]$. Έστω t_{e,α_e} η μικρότερη χρονική στιγμή t' τ.ω $\sum_{t=1}^{t'} x_{e,t} \geq \alpha_e$. Σχηματίζουμε την τελική μετάθεση διατάζοντας τα στοιχεία $e \in U$ σε φθίνουσα σειρά ως προς τα t_{e,α_e} με αυθαίρετο tie-breaking. Για περισσότερες λεπτομέρειες ως προς την ανάλυση του αλγορίθμου παραπέμπουμε στο original paper [10].

1.3 Το πρόβλημα ανανέωσης λίστας(List Update)

Πρωτού περάσουμε στην μελέτη της online εκδοχής του Min-Sum Set Cover παρουσιάζουμε ένα κλασικό online πρόβλημα το οποίο έχει μελετηθεί σε βάθος. Πέρα απ' το ότι το πρόβλημα List Update αποτελεί ένα απλό και εύπεπτο online πρόβλημα για εισαγωγή κινητοποίησε και σε μεγάλο βαθμό την γενίκευση του Min-Sum Set Cover ως προς την online και την offline-Multistage εκδοχή τους.

Το πρόβλημα List Update είναι ένα απ' τα κλασικά προβλήματα στον τομέα του Online Optimization. Μας δίνεται αρχικά μια μετάθεση του συνόλου $\{1, 2, ..., n\}$ και ένα σύνολο από requests $r_1, r_2, ..., r_m$ όπου $r_i \in \{1, 2, ..., n\}$. Για κάθε request r_i πληρώνουμε κάθε φορά του κόστος πρόσβασης που είναι η θέση του στοιχείου r_i στην τωρινή μετάθεση. Έπειτα, έχουμε την επιλογή να μεταφέρουμε το συγκεκριμένο στοιχείο πιο κοντά στην αρχή της λίστας προκειμένου να κάνουμε τα μελλοντικά request 'φθηνότερα'.

Στην online εκδοχή του προβλήματος όταν μας δίνεται ένα συγκεκριμένο request δεν γνωρίζουμε τίποτα για τα μελλοντικά request που θα έρθουν.

1.3.1 Ένας αφελής αλγόριθμος για το List Update

Μια φυσιολογική πρώτη απόπειρα για να λύσουμε το πρόβλημα θα ήταν να διατάξουμε την λίστα βάσει της τωρινής συχνότητας πρόσβασης των στοιχείων. Διαισθητικά αυτός ο αλγόριθμος φαίνεται λογικός καθώς αν θέλουμε να κρατήσουμε χαμηλό το κόστος βγάζει αρκετό νόημα τα στοιχεία που ζητούνται συχνά να βρίσκονται στην αρχή της λίστας. Επιπλέον, το να διατάξουμε την μετάθεση με αυτόν τον τρόπο είναι εφικτό στο μοντέλο του προβλήματος (όπου κάθε φορά μπορούμε να κουνήσουμε μόνο το στοιχείο που ζητήθηκε) καθώς όταν μας έρχεται ένα request r_i, η συχνότητά του αυξάνεται και συνεπώς θα θέλουμε να το μεταφέρουμε πιο μπροστά στην λίστα. Δυστυχώς, το ακόλουθο λήμμα μας δείχνει ότι αυτός ο αλγόριθμος οδηγεί σε τετριμμένα κακό competitive ratio.

Λήμμα 1. *Ο αλγόριθμος* Order By Frequency *είναι* $\Omega(n)$*-competitive.*

Απόδειξη. Ας υποθέσουμε ότι ξεκινάμε με την μετάθεση $[1, 2, 3, ..., n]$. Ας υποθέσουμε ότι μας έρχεται το ακόλουθο request sequence: Το στοιχείο 1 το ζητάμε n φορές, Το στοιχείο 2 το ζητάμε n φορές, ..., το στοιχείο n το ζητάμε n φορές. Τώρα, αν θεωρήσουμε το κόστος πρόσβασης για το δεύτερο μισό των στοιχείων, δηλαδή τα στοιχεία $n/2, n/2 + 1, ..., n$. Καθένα απ' αυτά τα στοιχεία έχουν κόστος τουλάχιστον $n/2$ καθώς τα πρώτα $n/2$ στοιχεία έχουν κόστος n. Συνεπώς το συνολικό κόστος του αλγορίθμου γι' αυτό το παράδειγμα είναι $\Omega(n^3)$.

Παρ' όλ' αυτάμ μπορούμε να δούμε ότι υπάρχει αλγόριθμος που πετυχαίνει συνολικό κόστος $O(n^2)$. Αν μετακινήσουμε κάθε φορά το ζητούμενο στοιχείο στην αρχή της λίστας βλέπουμε ότι το συνολικό κόστος είναι $O(n^2)$. Συνεπώς το competitive ratio του αλγορίθμου order by frequency algorithm είναι $\Omega(n)$. □

1.3.2 Ο αλγόριθμος Move-To-Front

Το παραπάνω λήμμα μας δίνει μια ωραία ιδέα για την κατασκευή ενός απλού και φυσιολογικού αλγορίθμου για το List Update. Μετακίνησε κάθε φορά το ζητούμενο στοιχείο στην αρχή της λίστας. Το εν λόγω αλγοριθμικό σχήμα που χρησιμοποιείται και στην offline εκδοχή του Min-Sum Set Cover καταφέρνει να πετύχει $O(1)$ competitiveness. Συγκεκριμένα, ο αλγόριθμος Move to Front(MTF) είναι 2-competitive για το List Update.

Lemma 1. *Ο αλγόριθμος MTF είναι 2-Competitive για το List Update.*

1.3.3 Competitive Analysis για τον αλγόριθμο MTF

Προκειμένου να αποδείξουμε το 2-competitiveness του αλγόριθμου MTF χρησιμοποιούμε την μέθοδο δυναμικού. Ορίζουμε συνάρτηση $\Phi : [0, m] \to \mathbb{Z}$ η οποία αποτυπώνει την κατάσταση της μετάθεσης μετά από κάθε request σε έναν ακέραιο αριθμό. Ο σκοπός μας είναι η συνάρτηση $\Phi(i)$ να εμπεριέχει πληροφορίες για την απόσταση της μετάθεσης που έχουμε δημιουργήσει απ' την βέλτιστη μετάθεση αφού εξυπηρετήσουμε το request r_i.

Έπειτα, ορίζουμε το amortized κόστος για κάθε βήμα βασιζόμενη στην συνάρτηση δυναμικού. Έστω $A(i) = MTF(i) + \Phi(i) - \Phi(i-1)$ όπου $MTF(i)$ είναι το κόστος που πληρώνει ο αλγόριθμος Move-To-Front για να εξυπηρετήσει το i-οστό request. Παρατηρούμε ότι:

$$\sum_{i=1}^{m} A(i) = \sum_{i=1}^{m} MTF(i) + \Phi(i) - \Phi(i-1) = \Phi(m) - \Phi(0) + \sum_{i=1}^{m} MTF(i) \qquad (1.1)$$

Συνεπώς, αν ορίζουμε την συνάρτηση δυναμικού με τέτοιο τρόπο ώστε $\Phi(m) - \Phi(0) > 0$ τότε το άθροισμα των amortized κοστών για όλα τα request που λαμβάνουμε είναι άνω φράγμα για το κόστος του αλγόριθμου MTF.

Πρωτού ορίσουμε την συνάρτηση δυναμικού, έστω $V(i)$ το σύνολο των ζευγών (x, y) τ.ω το x στην μετάθεση που διατηρείται απ' τον αλγόριθμο MTF είναι αντιστραμμένο σε σχέση με το y στην βέλτιστη μετάθεση αφού εξυπηρετηθεί το request i. Για να ξεκαθαρίσουμε τον ορισμό, έστω $Pos(\pi^i, x)$ η θέση του x στην μετάθεση π^i, δηλαδή $Pos(x) = (\pi^i)^{-1}(x)$. Έστω o^i η βέλτιστη μετάθεση την χρονική στιγμή i, δηλαδή μια οποιαδήποτε μετάθεση που διατηρείται απ' τον βέλτιστο offline αλγόριθμο αφού εξυπηρετηθεί το request i. Ορίζουμε:

$$V(i) = \{(x, y) \in [1, n] \times [1, n] \text{ s.t } Pos(\pi^i, x) < Pos(\pi^i, y) \wedge Pos(o^i, x) > Pos(o^i, y)$$

Συνεπώς το $V(i)$ είναι το σύνολο των ζευγών (x,y) τ.ω το x να έρχεται πριν το y στην μετάθεση που διατηρείται απ' τον αλγόριθμο MTF αλλά το x να έρχεται μετά το y στην βέλτιστη μετάθεση. Τέλος, ορίζουμε $\Phi(i) = |V(i)|$ και είμαστε έτοιμοι να αποδείξουμε το ακόλουθο λήμμα.

Λήμμα 2. *Έστω $A(i)$ το amortized κόστος όπως ορίστηκε παραπάνω. Έστω $OPT(i)$ το κόστος της βέλτιστης λύσης αφού εξυπηρετήσουμε το i-οστό request, τότε $A(i) \leq 2 \cdot OPT(i) - 1$.*

Απόδειξη. Έστω r_i το i-οστό request. Έστω επιπλέον k το πλήθος των στοιχείων που εμφανίζονται πριν το r_i τόσο στην μετάθεση του MTF όσο και στην βέλτιστη μετάθεση και έστω l το πλήθος των στοιχείων που εμφανίζονται πριν το r_i στην μετάθεση του MTF αλλά μετά το r_i στην βέλτιστη μετάθεση. Σύμφωνα με αυτούς τους ορισμούς έχουμε $MTF(i) = l + k + 1$ και επιπλέον $OPT(i) \geq k + 1$. Τώρα, παρατηρούμε ότι αν μετακινήσουμε το r_i στην αρχή της λίστας τότε δημιουργούμε k καινούργια inversions και καταστρέφουμε ακριβώς l inversions. Συνεπώς $\Phi(i) - \Phi(i-1) \leq k - l$. Συνδυάζοντας τις παραπάνω σχέσεις λαμβάνουμε:

$$A(i) = MTF(i) + \Phi(i) - \Phi(i-1) \leq l + k + 1 + (k - l) = 2 \cdot k + 1 \leq 2 \cdot (OPT(i) - 1) \leq 2 \cdot OPT(i) - 1$$

και η απόδειξη έχει ολοκληρωθεί. □

Τώρα είμαστε έτοιμοι να αποδείξουμε ότι ο αλγόριθμος MTF είναι 2-Competitive. Συνδυάζοντας τα παραπάνω λήμματα λαμβάνουμε $MTF = \sum_{i=1}^{m} MTF(i) \leq 2 * OPT - m + \Phi(0) - \Phi(m) \leq 2 * OPT - m + \binom{n}{2}$. Η τελευταία ανισότητα προκύπτει απ' το ότι $\Phi(0) = 0$ και $\Phi(i) \leq \binom{n}{2}$ αφού το πλήθος των αντιστροφών είναι πάντα μικρότερο ή ίσο από $\binom{n}{2}$.

1.3.4 Κάτω φράγμα στο Competitive Ratio

Στην παράγραφο αυτή αποδεικνύουμε ότι κάθε online αλγόριθμος για το πρόβλημα List Update είναι τουλάχιστον 2-competitive.

Απόδειξη. Για κάθε online αλγόριθμο κατασκευάζουμε ένα adversarial παράδειγμα εισόδου ως εξής: Θέτουμε r_i ίσο με το τελευταίο στοιχείο της τωρινής μετάθεσης. Προφανώς, για μια μετάθεση μεγέθους n και m συνολικά request το κόστος κάθε online αλγόριθμου είναι $n \cdot m$. Θα δείξουμε ωστόσο ότι υπάρχει ακολουθία από μεταθέσεις της οποίας το συνολικό κόστος είναι το πολύ $\frac{nm}{2}$.

Έστω μια ακολουθία από τυχαίες μεταθέσεις. Το αναμενόμενο κόστος για να εξυπηρετήσουμε το κάθε request r_i είναι $\frac{n}{2}$ και συνεπώς το συνολικό αναμενόμενο κόστος του αλγόριθμου είναι $\frac{nm}{2}$. Μπορούμε εύκολα να δούμε λοιπόν ότι θα πρέπει να υπάρχει ακολουθία μεταθέσεων με κόστος το πολύ ίσο με το αναμενόμενο και συνεπώς η απόδειξη έχει ολοκληρωθεί. $\square$

1.4 Η online έκδοση του Min Sum Set Cover

Πρωτού περάσουμε στον βασικό κορμό της παρούσας εργασίας παρουσιάζουμε την τελευταία σχετική παραλλαγή του προβλήματος που κινητοποίησε και την μελέτη του offline Multistage προβλήματος στο οποίο δουλέψαμε. Στα online προβλήματα, έχουμε μια κατάσταση η οποία εξελίσσεται στον χρόνο και οφείλουμε να απαντάμε ερωτήματα χωρίς να γνωρίζουμε εκ των προτέρων τι ερωτήματα θα μας έρθουν στο μέλλον (βλ. List Update). '

Στο [28] οι συγγραφείς εξερευνούν την Online εκδοχή του Min-Sum Set Cover. Παρατηρούν ότι παρόλο που το Online Min-Sum Set Cover αποτελεί γενίκευση του List Update(Στην ουσία το List Update είναι μια ειδική περίπτωση του Online Min-Sum Set Cover όπου όλα τα request έχουν μέγεθος 1, δηλάδη $|S_t| = 1$) ο αλγόριθμος Move-to-Front που αποτελεί τον κλασικό αλγόριθμο που πετυχαίνει το βέλτιστο competitive ratio για το List Update δεν δίνει ενδιαφέροντα αποτελέσματα. Στην συνέχεια αυτού του κεφαλαίου θεωρούμε πως τα request είναι r-ομοιόμορφα, δηλαδή $|S_t| = r$ για κάθε $t \in [T]$.

1.4.1 Ένας αλγόριθμος χωρίς μνήμη για το Online-Dynamic Min-Sum Set Cover

Το πρώτο αποτέλεσμα που παρουσιάζουμε απ' το αντίστοιχο paper είναι ένας αλγόριθμος χωρίς μνήμη για την δυναμική εκδοχή Online Min-Sum Set Cover. Ένας online αλγόριθμος λέγεται 'χωρίς μνήμη' ανν οι αποφάσεις που λαμβάνονται την χρονική στιγμή t δεν εξαρτώνται απ' τις προηγούμενες χρονικές στιγμές. Παρακάτω, παρουσιάζουμε τον αλγόριθμο Move-All-Equally(MAE). Επί της ουσίας αυτό που κάνουμε είναι πως όταν μας δωθεί κάποιο request S_t μετακινούμε κάθε στοιχείο του S_t στην τωρινή μετάθεση π^{t-1} προς τα αριστερά μέχρι να φτάσουν στην αρχή της μετάθεσης. Αναλυτικά ο αλγόριθμος φαίνεται παρακάτω.

Algorithm 2 Move-All-Equally

```
Input: The given request at time t $S_t$ and the permutation $\pi^{t-1}$.
Output: A permutation at time $t$ $\pi^t$.
1: $k_t \leftarrow min\{i | \pi^{t-1}[i] \in S_t\}$
2: decrease the index of all elements on $S_t$ by $k_t - 1$.
```

Ο παραπάνω αλγόριθμος αν και έχει ενδιαφέρουσες ιδιότητες φαίνεται να παρουσιάζει ένα gap μεταξύ άνω και κάτω ορίου ως προς το competitive ratio.

Lemma 2. *Το Competitive ratio του αλγορίθμου Move-All-Equally είναι* $\Omega(r^2)$.

Απόδειξη. Κατασκευάζουμε μια adversarial (κακόβουλη) ακολουθία εισόδου για να δείξουμε το κάτω όριο. Κάθε φορά φτιάχνουμε ένα request S_t που αποτελείται απ' τα τελευταία r στοιχεία της τωρινής μετάθεσης. Εφ' όσον ο αλγόριθμος μετακινεί τα στοιχεία με την ίδια ταχύτητα παρατηρούμε πως έχουμε n/r requests οπου μεταξύ τους δεν θα έχουν κοινά στοιχεία. Συνεπώς, για να καλύψουμε κάθε request βέλτιστα έχουμε κόστος $\Theta(n/r)$ ενώ ο αλγόριθμος MAE καταφέρνει να πετύχει κόστος $\Omega(n \cdot r)$. Συνεπώς, από σύγκριση των δύο λαμβάνουμε το κάτω όριο για το competitive ratio. $\square$

Αν και η ανάλυση του αλγορίθμου δεν μας δίνει κάποιο matching άνω όριο έχουμε το παρακάτω ενδιαφέρον αποτέλεσμα. Για περισσότερες λεπτομέρειες στην ανάλυση του αλγορίθμου ο αναγνώστης μπορεί να ανατρέξει στο [28].

Lemma 3. *Το Competitive Ratio του αλγορίθμου Move-All-Equally είναι το πολύ* $O(2^{\sqrt{(logn \cdot logr)}})$.

1.4.2 Ο Αλγόριθμος Lazy Rounding

Σε αυτή την ενότητα παρουσιάζουμε ένα κάτω φράγμα για κάθε ντετερμινιστικό αλγόριθμο για το Online Min-Sum Set Cover και επιπλέον παρουσιάζουμε τον αλγόριθμο Lazy Rounding που αποτελεί έναν ντετερμινιστικό αλγόριθμο που ταιριάζει σε competitive ratio στο εν λόγω κάτω φράγμα (πλην κάποιες σταθερές).

Theorem 1. *Κάθε ντετερμινιστικός αλγόριθμος για το Online Min-Sum Set Cover έχει competitive ratio τουλάχιστον* $(r+1)(1-\frac{1}{n+1})$.

Για την απόδειξη του θεωρήματος οι συγγραφείς χρησιμοποιούν ένα πιθανοτικό επιχείρημα μέσου όρου πολύ παρόμοιο με αυτό που παρουσιάσαμε για το κάτω φράγμα στο List-Update [40]. Σε κάθε βήμα ο adversary κατασκευάζει ένα request πάλι με τα τελευταία r στοιχεία της μετάθεσης και συνεπώς έχει access cost τουλάχιστον $(n-r+1)$. Με ένα απλό συνδυαστικό επιχείρημα δείχνουν ότι για κάθε request set S_t μεγέθους r και κάθε $i \in [n-r+1]$, το πλήθος των μεταθέσεων π με κόστος πρόσβασης $\pi(S_t) = i$ είναι $\binom{n-i}{r-1} \cdot r! \cdot (n-r)!$. Αθροίζοντας πάνω σε όλες τις μεταθέσεις και διαιρώντας με $n!$ για να πάρουμε το μέσο κόστος πρόσβασης λαμβάνουμε ότι η βέλτιστη ακολουθία έχει κόστος πρόσβαση το πολύ $\frac{n+1}{r+1}$ και συνεπώς το κόστος πρόσβασης είναι τουλάχιστον $(r+1)(1-\frac{1}{n+1})$. Για περισσότερες λεπτομέρειες ο αναγνώστης μπορεί να ανατρέξει στο paper [28].

Παρακάτω παρουσιάζουμε σε high-level τον αλγόριθμο Lazy Rounding.

1. Εφαρμόζουμε τον αλγόριθμο Multiplicative Weights Update (MWU) σαν black-box με learning rate $1/n^3$. Χρησιμοποιώντας βασικά αποτελέσματα από learning theory οι συγγραφείς δείχνουν ότι: $AccessCost(MWU) \leq \frac{5}{4} AccessCost(OPT)$.

2. Με την χρήση ενός online σχήματος rounding μετατρέπουμε κάθε τυχαιοκρατικό αλγόριθμο A σε ντετερμινιστικό, που συμβολίζουμε ως $Derand(A)$, με κόστος πρόσβασης το πολύ $2r \cdot \mathbb{R}[AccessCost(A)]$, χωρίς όμως καμία εγγύηση ως προς το κόστος μετακίνησης του $Derand(A)$.

3. Ο αλγόριθμος Lazy-Rounding είναι επί της ουσίας μια lazy εκδοχή του $Derand(MWU)$ (εφαρμόζουμε δηλαδή του online σχήμα rounding στον αλγόριθμο Multiplicative Weights Update Algorithm) που ανανεώνει την μετάθεση αν και μόνο αν η κατανομή του MWU έχει αλλάξει κατά πολύ. Ας ορίσουμε ένα χρονικό παράθυρο όπου ο αλγόριθμος Lazy-Rounding δεν αλλάζει την μετάθεση φάση. Οι συγγραφείς παρατηρούν ότι σε κάθε φάση:

   - $AccessCost(Lazy - Rounding) \leq 4r \cdot \mathbb{E}[AccessCost(MWU)]$.
   - $MovingCost(Lazy - Rounding) \leq \mathbb{E}[AccessCost(MWU)]$.

   Συνδυάζοντας τις παραπάνω ιδιότητες λαμβάνουμε:

$$Cost(Lazy - Rounding) \leq (4r+1)\mathbb{E}[AccessCost(MWU)] \leq (5r+2)Cost(OPT)$$

.

Theorem 2. *Ο ντετερμινιστικός αλγόριθμος Lazy-Rounding είναι (5r+2) competitive για την Online εκδοχή του Min-Sum Set Cover.*

1.5 Multistage Min-Sum Set Cover

Στις προηγούμενες ενότητες κάναμε μια ιστορική αναδρομή και καλύψαμε τις βασικές θεωρητικές έννοιες που απαιτούνται για την κατανόηση του προβλήματος. Το αντικείμενο της παρούσας εργασίας αποτελεί η δυναμική offline εκδοχή του Min-Sum Set Cover. Κατά την διάρκεια της υπόλοιπης εργασίας όσο παρουσιάζουμε τα βασικά μας αποτελέσματα αναφερόμαστε στο εν λόγω πρόβλημα ως Multistage Min-Sum Set Cover, Dynamic Min-Sum Set Cover ή Mult-MSSC.

1.5.1 Βασικές Έννοιες και Ορισμοί

Συμβολίζουμε με U το σύνολο όλων των στοιχείων e, έχουμε φυσικά ότι $|U| = n$. Συμβολίζουμε με π μια οποιαδήποτε μετάθεση των στοιχείων του U και ως π_i συμβολίζουμε το στοιχείο που βρίσκεται στην θέση i. Επιπλέον, συμβολίζουμε με $\text{Pos}(e, \pi)$ την θέση του στοιχείου $e \in U$ στην μετάθεση π.

Ορισμός 1 (Απόσταση Kendall-Tau). *Δεδομένων δύο μεταθέσεων π^A, π^B θεωρούμε πως ένα ζεύγος στοιχείων (e, e') είναι αντεστραμμένο αν και μόνο αν $\text{Pos}(e, \pi^A) > \text{Pos}(e', \pi^A)$ και $\text{Pos}(e, \pi^B) < \text{Pos}(e', \pi^B)$ ή αντίστροφα. Η απόσταση Kendall-Tau δύο μεταθέσεων π^A, π^B είναι το πλήθος των αντεστραμμένων ζευγών στοιχείων και συμβολίζεται ως $\text{d}_{\text{KT}}(\pi^A, \pi^B)$.*

Ορισμός 2 (Spearman' Footrule Distance). *Η FootRule distance δύο μεταθέσεων π^A, π^B ορίζεται ως $\text{d}_{\text{FR}}(\pi^A, \pi^B) = \sum_{e \in U} |\text{Pos}(e, \pi^A) - \text{Pos}(e, \pi^B)|$.*

Παρατηρούμε πως η απόσταση Kendall-Tau και η FootRule distance είναι κατά προσέγγιση ισοδύναμες. Συγκεκριμένα, $\text{d}_{\text{KT}}(\pi^A, \pi^B) \le \text{d}_{\text{FR}}(\pi^A, \pi^B) \le 2 \cdot \text{d}_{\text{KT}}(\pi^A, \pi^B)$. Επιπλέον ικανοποιούν την τριγωνική ανισότητα.

Ορισμός 3. *Ένας πίνακας $n \times n$ με θετικά στοιχεία (όπου οι γραμμές αντιπροσωπεύουν τα στοιχεία και οι στήλες τις θέσεις) ονομάζεται στοχαστικός αν και μόνο αν $\sum_{i=1}^{n} A_{ei} = 1$ για κάθε $e \in U$ και διπλά στοχαστικός αν επιπλέον έχουμε $\sum_{e \in U} A_{ei} = 1$ for all $1 \le i \le n$.*

Η έννοια της απόστασης FootRule μπορεί να επεκταθεί και σε στοχαστικούς πίνακες. Δεδομένων δύο διπλα στοχαστικών πινάκων A, B θεωρούμε το πρόβλημα της μεταφοράς ελάχιστου κόστους, όπου θέλουμε να μετατρέψουμε την γραμμή A_e στην γραμμή B_e όπου η μεταφορά μιας μονάδας μάζας από την στήλη i στην στήλη j έχει κόστος $|i - j|$.

Ορισμός 4. *Η απόσταση FootRule μεταξύ δύο στοχαστικών πινάκων A, B την οποία συμβολίζουμε ως $\text{d}_{\text{FR}}(A, B)$, είναι η βέλτιστη τιμή του παρακάτω γραμμικού προγράμματος,*

$$\begin{array}{lll} min & \displaystyle\sum_{e \in U} \sum_{i=1}^{n} \sum_{j=1}^{n} |i - j| \cdot f_{ij}^{e} & \\ s.t & \displaystyle\sum_{i=1}^{n} f_{ij}^{e} = B_{ej} & \textit{για κάθε } e \in U \textit{ kai } j = 1, \dots, n \\ & \displaystyle\sum_{j=1}^{n} f_{ij}^{e} = A_{ei} & \textit{για κάθε } e \in U \textit{ και } i = 1, \dots, n \\ & f_{ij}^{e} \ge 0 & \textit{για κάθε } e \in U \textit{ και } i, j = 1, \dots, n \end{array}$$

Πλέον είμαστε σε θέση να ορίσουμε το πρόβλημα της Δυναμικής Κάλυψης Συνόλου Ελάχιστου Αθροίσματος αυστηρά.

Ορισμός 5 (Multistage Min-Sum Set Cover). *Δεδομένου ενός συνόλου U, μιας ακολουθίας από requests $R_1, \ldots, R_T \subseteq U$ και μιας αρχικής μετάθεσης των συνόλων του U, π^0 ο στόχος μας είναι να επιλέξουμε μια ακολουθία από μεταθέσεις $\pi^1, \ldots, \pi^T$ οι οποία ελαχιστοποιεί την ακόλουθη ποσότητα,*

$$\sum_{t=1}^{T} \pi^t(R_t) + \sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})$$

όπου $\pi^t(R_t)$ είναι η θέση του πρώτου στοιχείου του συνόλου R_t που συναντάμε στην μετάθεση π^t, $\pi^t(R_t) = \min\{1 \le i \le n : \pi_i^t \in R_t\}$.

Στην συνέχεια, θα αναφερόμαστε στην ποσότητα $\sum_{t=1}^{T} \pi^t(R_t)$ ως **κόστος κάλυψης** και στην ποσότητα $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})$ ως **κόστος μετακίνησης**. Συμβολίζουμε με π^t_{Opt} την μετάθεση της βέλτιστης λύσης για το Mult-MSSC την χρονική στιγμή t, με o_t το στοιχείο που η βέλτιστη λύση χρησιμοποιεί για να καλύψει το request R_t (δηλαδή το στοιχείο του R_t που εμφανίζεται πρώτο στην μετάθεση π^t_{Opt}), και με $\mathrm{OPT}_{\text{Mult-MSSC}}$ το κόστος της βέλτιστης λύσης. Τέλος, θα λέμε πως ένα instance του Mult-MSSC είναι *r-φραγμένο* στην περίπτωση που η πληθικότητα όλων των request είναι φραγμένη από r, $|R_t| \le r$.

1.5.2 Αποτελέσματα περί δυσκολίας του Dynamic Min-Sum Set Cover

Πρωτού περάσουμε στην παρουσίαση των βασικών αλγορίθμων και των αποτελεσμάτων μας παρουσιάζουμε ένα βασικό θεώρημα με το οποίο αποδεικνύεται πως το Mult-MSSC είναι τουλάχιστον όσο δύσκολο και το πρόβλημα της κάλυψης συνόλου. Συγκεκριμένα, υπάρχει μια αναγωγή που είναι approximation-preserving απ' το Set − Cover στο Mult-MSSC που μας δίνει τα ακόλουθα αποτελέσματα μη-προσεγγισιμότητας.

Θεώρημα 3.

- *Δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου με παράγοντα προσέγγισης $o(logn)$ για το* Mult-MSSC *εκτός και αν* P = NP.
- *Για r-φραγμένες ακολουθίες, δεν υπάρχει αλγόριθμος πολυωνυμικού χρόνου με παράγοντα προσέγγισης $o(r)$ για το* Mult-MSSC, *εκτός και αν υπάρχει αλγόριθμος με παράγοντα προσέγγισης $o(r)$ για το* Set − Cover *όπου κάθε στοιχείο καλύπτεται απ' το πολύ r σύνολα.*

Η απόδειξη του παραπάνω θεωρήματος είναι σχετικά απλή, δεδομένου ενός instance του Set − Cover κατασκευάζουμε ένα instance για το Mult-MSSC όπου η αρχική μετάθεση π^0 εμπεριέχει στις πρώτες θέσεις dummy στοιχεία (δηλαδή στοιχεία που δεν εμφανίζονται σε κανένα request) και στις τελευταίες θέσεις τα σύνολα του Set − Cover (θεωρούμε ένα στοιχείο του Mult-MSSC για κάθε σύνολο του Set − Cover). Η απόδειξη αναλυτικά παρουσιάζεται στο αγγλικό κείμενο της εν λόγω διπλωματικής εργασίας.

1.6 Αλγόριθμοι Προσέγγισης για το Dynamic Min-Sum Set Cover

Στην συνέχεια παρουσιάζουμε 2 προσεγγιστικούς αλγορίθμους που αποτελούν και τον βασικό κορμό αυτής της εργασίας. Ο πρώτος εξ' αυτών αποτελεί έναν πολυωνυμικό τυχαιοκρατικό αλγόριθμο με παράγοντα προσέγγισης $O(\log^2 n)$ για την γενική εκδοχή του προβλήματος, ενώ ο δεύτερος έχει παράγοντα προσέγγισης $O(r^2)$ για r-φραγμένες ακολουθίες από requests.

Και οι δύο αλγόριθμοι βασίζονται σε τεχνικές rounding ενός γραμμικού προγράμματος που ονομάζουμε *Fractional Move To Front* το οποίο αποτελεί LP Relaxation του *Move To Front*, ένα πρόβλημα στενά συνδεδεμένο με το Mult-MSSC. Για περισσότερες πληροφορίες ο αναγνώστης μπορεί να ανατρέξει στις ενότητες του αγγλικού κειμένου όπου καλύπτονται οι έννοιες του γραμμικού προγραμματισμού καθώς και του List Update.

Στο πρόβλημα Move-to-Front(για συντομία MTF) ψάχνουμε να βρούμε μια ακολουθία από μεταθέσεις $\pi^1, \ldots, \pi^T$ τέτοια ώστε σε κάθε 'γύρο' t, επιλέγουμε ένα στοιχείο του συνόλου R_t το οποίο τοποθετούμε στην αρχή της μετάθεσης π^t. Σκοπός μας είναι να ελαχιστοποιήσουμε το άθροισμα $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\pi^t, \pi^{t-1})$.

Ορισμός 6. *Δεδομένης μιας ακολουθίας από request $R_1, \ldots, R_T \subseteq U$ και μια αρχική μετάθεση π^0, θεωρούμε το ακόλουθο γραμμικό πρόγραμμα που ονομάζουμε* Fractional − MTF*,*

$$\begin{aligned}
min \quad & \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1}) \\
s.t \quad & \sum_{i=1}^{n} A^t_{ei} = 1 \quad \text{για κάθε } e \in U \text{ και } t = 1, \ldots, T \\
& \sum_{e \in U} A^t_{ei} = 1 \quad \text{για κάθε } i = 1, \ldots, n \text{ και } t = 1, \ldots, T \\
& \sum_{e \in R_t} A^t_{e1} = 1 \quad \text{για κάθε } t = 1, \ldots, T \\
& A^0 = \pi^0 \\
& A^t_{ei} \geq 0 \qquad \text{για κάθε } e \in U,\ i = 1, \ldots, n \text{ και } t = 1, \ldots, T
\end{aligned}$$

όπου $\mathrm{d}_{\mathrm{FR}}(\cdot,\cdot)$ είναι το Footrule distance.

Υπάρχει ένα κομψό επιχείρημα (που έχει εμφανιστεί προηγουμένως και σε άλλες εργασίες π.χ, [28]) που δείχνει πως η βέλτιστη λύση του MTF είναι το πολύ $4 \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$. Στο παρακάτω λήμμα παρουσιάζουμε το επιχείρημα πως το Fractional − MoveToFront είναι ενα relaxation του Mult-MSSC με παράγοντα προσέγγισης 4.

Λήμμα 3. $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1}) \leq 4 \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$ *όπου $A^1, \ldots, A^t$ is the optimal solution of* Fractional − MTF.

1.6.1 Τυχαιοκρατικός Αλγόριθμος με Παράγοντα Προσέγγισης $O(log^2 n)$

Όπως έχουμε ήδη αναφέρει, οι βασικές συνεισφορές αυτής της εργασίας είναι ο σχεδιασμός ορισμένων *rounding schemes* όπου μετασχηματίζουν την βέλτιστη λύση, $A^1, \ldots, A^T$, του γραμμικού προγράμματος Fractional − MTF σε μια ακολουθία από μεταθέσεις $\pi^1, \ldots, \pi^T$. Αυτό γίνεται ώστε να μπορέσουμε να φράξουμε το κόστος μετακίνησης των αλγορίθμων μας από το κόστος μετακίνησης $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$. Έπειτα, μπορούμε ξεχωριστά να φράξουμε το κόστος κάλυψης, $\sum_{t=1}^{T} \pi^t(R_t)$ δείχνοντας πως πάντα ένα στοιχείο του R_t βρίσκεται στις πρώτες θέσεις της μετάθεσης π^t.

Η βασική τεχνική δυσκολία στον σχεδιασμό των εν λόγω rounding schemes είναι πως πρέπει να πιστοποιήσουμε και να αποδείξουμε ότι το κόστος μετακίνησης των λύσεών μας $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})$ φράζεται κατά προσέγγιση από το κόστος μετακίνησης $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$. Παρόλο που η σύνδεση μεταξύ διπλά στοχαστικών πινάκων και μεταθέσεων έχει μελετηθεί πολύ και υπάρχουν διάφορα rounding schemes όπου μετατρέπουν διπλά στοχαστικούς πίνακες σε κατανομές πιθανότητας πάνω σε μεταθέσεις (όπως το Birkhoff–von Neumann decomposition η τα σχήματα των [10, 39, 8, 28]), η χρήση τέτοιων αλγοριθμικών σχημάτων ως μάυρα κουτιά δεν φαίνεται να δίνει κάποιο θετικό αποτέλεσμα για το Mult-MSSC. Για παράδειγμα στην περίπτωση όπου $A^1 = \cdots = A^T$ και συνεπώς $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1}) = \mathrm{d}_{\mathrm{FR}}(A^1, A^0)$. Σε περίπτωση όπου ένας τυχαιοκρατικός αλγόριθμος εφαρμοστεί ανεξάρτητα σε κάθε A^t, υπάρχει πάντα η πιθανότητα να έχουμε $\pi^t \neq \pi^{t-1}$ και συνεπώς το κόστος μετακίνησης θα αποκλείνει απεριόριστα απ' το $\mathrm{d}_{\mathrm{FR}}(A^1, A^0)$ καθώς μεγαλώνει το T. Αυτό μας οδηγεί στο να χρησιμοποιήσουμε κάποιου είδους coupling όπου μετατρέπει την ακολουθία των πινάκων $A^1, \ldots, A^T$ σε ακολουθία μεταθέσεων $\pi^1, \ldots, \pi^T$. Ο πρώτος αλγόριθμος που παρουσιάζουμε στην συνέχεια χρησιμοποιεί μια τέτοια μέθοδο και αποτελεί και την βασική συνεισφορά της εργασίας μας για request με αυθαίρετο πλήθος στοιχείων.

Algorithm 3 Τυχαριοκρατικός Αλγόριθμος για το πρόβλημα Mult-MSSC

```
Input: Μια ακολουθία από request $R_1, \ldots, R_T$ και μια αρχική μετάθεση των στοιχείων $\pi^0$.
Output: Μια ακολουθία μεταθέσεων $\pi^1, \ldots, \pi^T$.
 1: Find the optimal solution $A^0 = \pi^0, A^1, \ldots, A^T$ for Fractional − MTF.
 2: for each element $e \in U$ do
 3:     Select $\alpha_e$ uniformly at random in $[0, 1]$.
 4: end for
 5: for $t = 1 \ldots T$ do
 6:     for all elements $e \in U$ do
 7:         $I_e^t := \mathrm{argmin}_{1 \le i \le n}\{\log n \cdot \sum_{s=1}^{i} A_{es}^t \ge \alpha_e\}$.
 8:     end for
 9:     $\pi^t :=$ sort elements according to $I_e^t$ with ties being broken lexicographically.
10: end for
```

Το rounding scheme του παραπάνω αλγορίθμου, θέτει το coupling που αναφέραμε μεταξύ διαφορετικών χρονικών στιγμών με το να απαιτεί κάθε στοιχείο e να διαλέξει ένα α_e μια φορά στην αρχή και για όλη την διάρκεια του αλγορίθμου καθώς και με την επιλογή να σπάει τις ισοπαλίες με αλφαβητική σειρά (οποιοσδήποτε συνεπής κανόνας για τις ισοπαλίες θα δούλευε εξίσου καλά).

Μπορούμε να δείξουμε ότι ανεξάρτητα απ την ακολουθία των διπλά στοχαστικών πινάκων που λαμβάνουμε ως είσοδο ο αλγόριθμος μας παράγει μια ακολουθία μεταθέσεων με αναμενόμενο συνολικό κόστος μετακίνησης το πολύ $4 \log^2 n$ φορές το κόστος μετακίνησης της ακολουθίας πινάκων και συνεπώς το συνολικό κόστος μετακίνησης του αλγορίθμου φράζεται από $4 \log^2 n \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$.

Θεώρημα 4. *Ο παραπάνω αλγόριθμος έχει λόγο προσέγγισης $O(\log^2 n)$ για το πρόβλημα* Mult-MSSC.

Παρότι στο βήμα 7 του αλγορίθμου πολλαπλασιάζουμε τα στοιχεία του πίνακα A^t με $\log n$ ο λόγος προσέγγισης είναι $O(\log^2 n)$. Εκ πρώτης όψης αυτό φαίνεται περίεργο αλλά η εξήγηση προκύπτει πολύ φυσιολογικά. Για τις περισσότερες θέσεις i, η πιθανότητα ότι ένα στοιχείο e θα έχει εν τέλει index $I_e^t = i$ είναι περίπου $\log n \cdot A_{ei}^t$, αλλά επειδή κάθε index $j \le i$ κατά μέση τιμή επιλέγεται από άλλα $\log n$ στοιχεία, η αναμενόμενη θέση του στοιχείου e στην τελική μετάθεση θα είναι περίπου $\log^2 n$ φορές η μέση τιμή του $\mathrm{argmin}_{1 \le i \le n}\{\sum_{s=1}^{i} A_{es}^t \ge \alpha_e\}$. Αυτό το φαινόμενο έρχεται σε αντιστοιχία με το επιχείρημα στην εργασία [24] όπου αποδεικνύεται για την στατική εκδοχή του προβλήματος ότι ο άπληστος αλγόριθμος έχει παράγοντα προσέγγισης 4 για το πρόβλημα *Min-Sum Set Cover*

1.6.2 Ντετερμινιστικος Αλγόριθμος για r-φραγμένες Ακολουθίες

Συνεχίζουμε με τον δεύτερο αλγόριθμο που αφορά r-φραγμένες ακολουθίες από request. Γνωρίζουμε ότι στο πρόβλημα Set − Cover υπάρχουν διάφορα αλγοριθμικά σχήματα βασισμένα σε γραμμικό προγραμματισμό που πετυχαίνουν παράγοντα προσέγγισης r, όπου κάθε στοιχείο μπορεί να ανήκει το πολύ σε r σύνολα. Θέλουμε να δούμε επομένως κατά πόσο μπορούμε να κάνουμε κάτι καλύτερο για το Mult-MSSC από $O(\log^2 n)$ στην περίπτωση που έχουμε r-φραγμένη ακολουθία από request.

Ο απλός άπληστος αλγόριθμος που παρουσιάζουμε παρακάτω μας δίνει παράγοντα προσέγγισης $O(r^2)$ σε αυτή την περίπτωση.

Algorithm 4 Ένας άπληστος αλγόριθμος για το Mult-MSSC για r-φραγμένες ακολουθίες.

```
Είσοδος: Μια ακολουθία από request $R_1, \dots, R_T$ with $|R_t| \leq r$ και μια αρχική μετάθεση $\pi^0$.
Output: Μια ακολουθία μεταθέσεων $\pi^1, \dots, \pi^T$.
1: Find the optimal solution $A^0 = \pi^0, A^1, \dots, A^T$ for Fractional − MTF.
2: for $t = 1 \dots T$ do
3:     $\pi^t :=$ in $\pi^{t-1}$, move to the first position an element $e \in R_t$ such that $A^t_{e1} \geq 1/r$
4: end for
```

Ο παράγοντας προσέγγισης $O(r^2)$ αποδεικνύεται πλήρως στην συνέχεια της διπλωματικής εργασίας στο θεώρημα 10. Η βασική τεχνική δυσκολία είναι πως δεν μπορούμε να συγκρίνουμε απευθείας το κόσοτος του αλγορίθμου μας με τον στόχο $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$ και συνεπώς εφαρμόζουμε μια άλλη τεχνική.

Στο πρώτο βήμα (λήμμα 15), αποδεικνύουμε την ύπαρξη μιας ακολουθίας από διπλά στοχαστικούς πίνακες $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ όπου ο καθένας εκ των οποίων $\hat{A}^t$ ικανοποιεί τα ακόλουθα κριτήρια,

1. Τα στοιχεία του είναι **πολλαπλάσια του** $1/r$
2. $\hat{A}^t_{e_t 1} \geq 1/r$ όπου e_t είναι το στοιχείο που ο αλγόριθμος μεταφέρει στην πρώτη θέση την χρονική στιγμή t.
3. Η ακολουθία $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ έχει κόστος το πολύ $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$.

Προκειμένου να δείξουμε πως μια τέτοια ακολουθία υπάρχει κατασκευάζουμε ένα κατάλληλο γραμμικό πρόγραμμα (βλέπε ορισμό 14) βασισμένοι στα στοιχεία όπου ο αλγοριθμός μας μεταφέρει στην πρώτη θέση και αποδεικνύουμε πως έχει βέλτιστη λύση όπου όλες οι μεταβλητές παίρνουν τιμές που είναι πολλαπλάσια του $1/r$. Προκειμένου να το κάνουμε αυτό, συσχετίζουμε το γραμμικό πρόγραμμα του ορισμού 14 με μια κλασματική εκδοχή του προβλήματος k-Paging [9] και βασισμένοι στην βέλτιστη πολιτική εκδίωξης στοιχείων (*Διώξε το στοιχείο που εμφανίζεται αργότερα ξανά στο μέλλον*). Σχεδιάζουμε λοιπόν έναν αλγόριθμο που παράγει βέλτιστες λύσεις για το γραμμικό πρόγραμμα με τιμές πολλαπλάσια του $1/r$.

Στο δεύτερο βήμα (λήμμα 16), δείχνουμε ότι για κάθε ακολουθία πινάκων $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ που ικανοποιούν τις ιδιότητες που αναφέραμε παραπάνω, το συνολικό κόστος κίνησης του άπληστου αλγορίθμου είναι το πολύ $O(r^2) \cdot \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1})$. Αυτό το δείχνουμε με την χρήση κατάλληλης συνάρτησης δυναμικού βασισμένη σε μια γενίκευση της απόστασης Kendall-Tau (βλέπε ορισμό 18).

Θεώρημα 5. *Ο άπληστος αλγόριθμος για r-φραγμένες ακολουθίες request έχει παράγοντα προσέγγισης $O(r^2)$ για το* Mult-MSSC.

Κείμενο στα αγγλικά

Chapter 1

Introduction

In this Diploma Thesis we investigate the polynomial time approximability of Multistage Min-Sum Set Cover. Multistage Min-Sum Set Cover (or Dynamic Min-Sum Set Cover) is a natural and interesting generalization of List Update and it falls in the general category of dynamic problems that evolve over time. In this particular problem we are given a universe U and a collection of sets $S_1, S_2, ..., S_T$ and our goal is to produce a set of permutations $\pi^1, \pi^2, \dots, \pi^T$ so that we minimize a specific cost function. We will define the problem rigorously further on as well as relative problems that fall into the same category.

Even though several variations of the problem have been extensively studied such as the Static [23] or the Generalized version [7, 10] of Min-Sum Set Cover it is the first time that the Multistage offline version is considered. Our work is heavily motivated by the recent work at [28] where the authors study the competitive ratio of the Online Min-Sum Set Cover problem. Even though several results are proven to be tight there are still gaps to be closed. By studying the offline-Multistage version of the problems our goals are to match the lower bounds of the problem with appropriate algorithms and motivate the further study of the online version using some of the techniques used in this work.

We begin with an investigation of the classical well-studied problems that appear in the literature and are generalized by MSSC. We aim to present the state of the art algorithms used and the techniques used for analysing them. In many cases, our algorithms for the Multistage version are heavily influenced by algorithms that appeared previously for other (often unrelated) versions of the problems.

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 [24] proved that the greedy algorithm, which picks in each position the element that covers the most uncovered requests, is a 4-approximation (that was also implicit in [11]) and that no $(4 - \varepsilon)$-approximation is possible, unless $\text{P} = \text{NP}$. In Generalized MSSC (a.k.a. *Multiple Intents Re-ranking*), there is a covering requirement $K(R_t)$ for each request R_t and the cost of covering a request R_t is the position of the $K(R_t)$-th element of R_t in the (static) permutation π. The MSSC problem is the special case where $K(R_t) = 1$ for all requests R_t. Another notable special case of Generalized MSSC is the Min-Latency Set Cover problem [31], which corresponds to the other extreme case where $K(R_t) = |R_t|$ for all requests R_t. Generalized MSSC was first studied by Azar et al. [7], who presented a $O(\log r)$-approximation; later $O(1)$-approximation algorithms were obtained [10, 39, 34, 8].

Further generalizations of Generalized MSSC have been considered, such as the Submodular Ranking problem, studied in [6], which generalizes both Set Cover and MSSC, and the Min-Latency Submodular Cover, studied by Im et al. [33]. We refer to [33, 32] for a detailed discussion on the connections between these problems and their applications.

The online version of MSSC, which generalizes the famous List Update problem, was studied in [28]. They proved that its static deterministic competitive ratio is $\Theta(r)$ and presented a natural memoryless algorithm, called *Move-all-Equally*, with static competitive ratio in $\Omega(r^2)$ and $2^{O(\sqrt{\log n \cdot \log r})}$

and dynamic competitive ratio in $\Omega(r\sqrt{n})$ and $O(r^{3/2}\sqrt{n})$-competitive. Subsequently, [29] considered MSSC from the viewpoint of online learning. Through dimensionality reduction from permutations to doubly stochastic matrices, they obtained randomized (resp. deterministic) polynomial-time online learning algorithms with $O(1)$-regret for Generalized MSSC (resp. $O(r)$-regret for MSSC).

1.1 The Static version of Min Sum Set Cover

We begin our investigation on the static version of Min Sum Set Cover. The input to the min sum set cover problem is a collection of n sets that jointly cover m elements. The output is a linear order on the sets, namely, in every time step from 1 to n exactly one set is chosen. For every element, this induces a first time step by which it is covered. The objective is to find a linear arrangement of the sets that minimizes the sum of these first time steps over all elements.

Formally, we define the Min Sum Set Cover problem as follows. Consider a set of elements U with $|U| = n$, without loss of generality we will suppose that $U = \{1, 2, ..., n\} = [1, n]$. Additionally, suppose we have m sets $S_1, S_2, ..., S_m$. Our objective is to find a permutation of the n elements that minimize the sum of costs of all the m sets S_i, that is $\sum_{i=1}^{m} C(S_i)$ where the cost of each set $C(S_i)$ is the position of the element of the set that appears first in the permutation.

The Min Sum Set Cover problem has been extensively studied. Up next we show the describe the main algorithm as well as the core results. For more details the reader can refer to [24].

1.1.1 The Greedy Algorithm

The approximation algorithm for Min Sum Set Cover is fairly natural.

Algorithm 5 Greedy Algorithm for Min Sum Set Cover

Input: A universe U as well as a collection of m sets $S_1, S_2, ..., S_m$
Output: A permutation of the elements of U.

```
1: $\pi = [\,]$
2: while $\pi$ doesn't contain all the elements of $U$ do
3:     Append to the end of $\pi$ the element of U that is contained in the most sets.
4: end while
5: return $\pi$
```

Even though the algorithm above is very simple the authors of [24] provide us with the following powerful results.

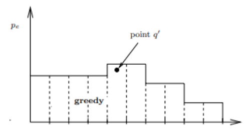
Theorem 3. *Algorithm 5 approximates min sum set cover with a ratio no worse than 4.*

Proof. Let **opt** denote the optimal solution for Min-Sum Set Cover and let **greedy** denote the value returned by Algorithm 5.

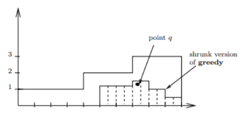
For $i = 1, 2, \ldots, n$ let X_i denote the number of sets covered by Algorithm 5 for the first time at step i. Let $R_i = [1, m] \setminus \bigcup_{j=1}^{i-1} X_i$, that is the sets not covered prior to step i. Observe that **greedy** $= \sum_{i=1}^{n} i * |X_i|$ and equivalently **greedy** $= \sum_{i=1}^{n} |R_i|$.

For $1 \leq i \leq n$ we define $P_i = \frac{|R_i|}{|X_i|}$. For every set $S \in X_i$ we define $p_S = P_i$ and also **price** $= \sum_S p_S$. Summing over all sets: **price** $= \sum i = 1^n |X_i| \cdot P_i = \sum_{i=1}^{n} |X_i| \cdot \frac{|R_i|}{|X_i|} = \sum_{i=1}^{n} |R_i| =$ **greedy**.

Lemma 4. *For the assignment of price given above we have:* ***opt*** $\geq$ ***price***$/4$.

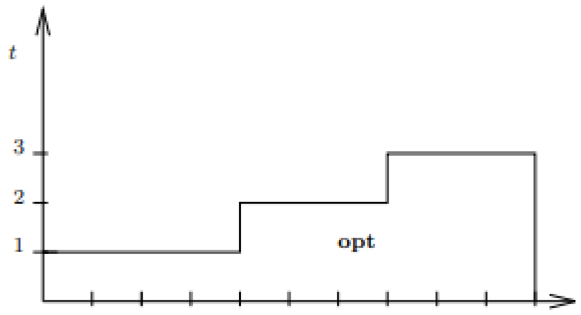


(a) The greedy histogram before shrinking



(b) Shrinking and Alignment

Proof. We consider the construction of the following histogram. For every set S_i we add a column to the histogram whose height is equal to the time step at which it was first covered in the optimal algorithm. Therefore we obtain a histogram whose total area is equal to the optimal solution **opt**.



Now we construct another histogram for the greedy solution. We have m columns, one for every set S_i. We order the columns (i.e sets) in the order they are covered by the greedy algorithm but the height of each column, rather than the timestep at which it was covered is now the price, as defined previously. The total area of the histogram is exactly equal to **price** $=$ **greedy**.

What we want to show is that **price** is at most 4 times **opt**. The way we show that is by shrinking the second histogram by a factor of 4 and show that it fits completely inside the optimal histogram. The way we do that is by shrinking the height and the width by a factor of 2. Therefore, for each column we have a height of $p_i/2$ and the width of the histogram is now $m/2$.

Now, we align the greedy histogram to the right of the optimal histogram so that it occupies the space from $m/2+1$ up to m on the horizontal axis. Now we claim that the shrunk greedy histogram is completely inside the optimal histogram and therefore this proves Lemma 4.

Consider an arbitrary point q' in the original (unshrunk) greedy histogram and let S be the set it corresponds to. Let i be the time step at which it was covered, then the height of this column is at most $\frac{|R_i|}{|X_i|}$ and the distance of q' from the right boundary of the optimal histogram is at most R_i. Now by shrinking we map the point q' to a new point q. The height of this point is now $h \leq \frac{|R_i|}{2 \cdot |X_i|}$ and the its distance from the right boundary is at most $r \leq \frac{|R_i|}{2}$.

For our point q to lie within the optimal histogram it suffices to show that at timestep h at least r sets remain uncovered by the optimal algorithm. Consider the sets on R_i. No number can ever cover more than $|X_i|$ sets since our greedy algorithm selects at every time step the number that covers the most sets (and that is $|X_i$ at time step i). Therefore in $\lfloor h \rfloor$ steps the optimal algorithm could cover at most $\lfloor h \rfloor \cdot \lfloor |X_i| \rfloor \leq \frac{|R_i|}{2}$ sets and therefore leave at least $|R_i| - \frac{|R_i|}{2} \geq \lceil r \rceil$ sets uncovered which proves our proposition that q lies within the optimal histogram. □

By the proof of Lemma 4 we obtain that **opt** $\geq$ **price**$/4$ and since **price** $=$ **greedy** we have the proof of our original lemma.

□

A natural question is whether the approximation ratio of this algorithm can be improved. To answer that we present the following hardness result.

Theorem 4. *For every $\epsilon > 0$ it is NP-Hard to approximate min sum set cover within a ratio of* $4 - \epsilon$.

The authors of [24] continue by proving equivalent results for constrained versions of the problem as well as for the Min-Sum Vertex Cover, in which essentially every set S_i has a cardinality of 2. Even though the algorithms they propose build an interesting intuition regarding the problem as we move on to generalized versions of the problem we observe that purely combinatorial tools are not sufficient to tackle the problem. This observation leads to the use of Linear Programming Relaxation methods which we use extensively while deriving our own results. To get a better idea of an Linear Programming approach we study a natural Generalization of the Min-Sum Set Cover problem [39] and a randomized rounding schema that highlights several techniques.

1.2 The Generalized Min-Sum Set Cover

After presenting the static version of Min Sum Set Cover we proceed with the generalization of the problem which was first introduced by Azar, Gamzu and Yin in [7]. In this problem we are once again given a universe U with $|U| = n$, a collection of subsets of U, $S_1, S_2, ..., S_m \subseteq U$ and also an integer number for each set in the collection $K(S_i)$. Our purpose is to compute a permutation of U in order to minimize the covering cost of each set: $\sum_{i=1}^{m} C(S_i)$ where $C(S_i)$ is the position of the $K(S_i)$-th element of S_i that appears on the permutation.

In [7] the authors present an $O(logr)$-approximation algorithm where $r = max_i\{|S_i|\}$. Subsequently, a constant factor approximation algorithm was given [10] based on Linear Programming.

In this section we present the main results of [39] where the authors use a method called α-point scheduling which helps them round the Linear Programming Relaxation of the problem and they obtain an algorithm that has a significantly better approximation ratio than [10]. The following results and techniques are rather important since several ideas we used for extracting our own results for Mult-MSSC are based on [39].

The first step is solving the following Linear Programming Relaxation of the problem. We define the variables $y_{i,t}$ which we want to be 1 if and only if $C(S_i) < t$ (that is, the $K(S_i)$-th element of the set S_i appears before index t) and additionally the variables $x_{e,t}$ which we want to equal 1 if and only if the element e $e \in U$ appears at position t. Of course these hold for the integer program and since we solve a relaxation these variables will probably not be exactly equal to 1. Based on the constraints above we have the following linear program.

$$\begin{aligned}
\text{m}in \quad & \sum_{t=1}^{n}\sum_{i=1}^{m}(1 - y_{i,t}) \\
\text{s.t} \quad & \sum_{e\in U} x_{e,t} = 1 \quad \text{για κάθε } t \leq n \\
& \sum_{t=1}^{n} x_{e,t} = 1 \quad \text{για κάθε } e \in U \\
& \sum_{e\in S\setminus A}\sum_{t'<t} x_{e,t} \geq (K(S_i) - |A|)\cdot y_{i,t} \quad \text{για κάθε } i \leq m, A \subseteq S_i, t \leq n \\
& x_{e,t}, y_{i,t} \in [0,1] \quad \text{για κάθε } e \in U, i \leq m, t \leq n
\end{aligned}$$

The Linear Program at first glance seems to have exponentially many constraints, since we create a constraint for every subset of every set S_i. It was shown in [10] that this problem can be dealt with and the exponentially many constraints can be separated in polynomial time so that we can solve the LP efficiently.

The basic algorithm for solving the problem with 28 appriximation ratio is based on α point scheduling. Specifically, for every $e \in U$ we select independently and uniformly $\alpha_e \in [0,1]$. Let t_{e,α_e} be the first time t' s.t $\sum_{t=1}^{t'} x_{e,t} \geq \alpha_e$. We form the final permutation by ordering the elements $e \in U$ in decreasing order of t_{e,α_e} by breaking ties arbitrarily (but consistently). For more details on the analysis of the algorithm the interested reader may refer to [10].

1.3 The Online Version of Min Sum Set Cover

Before we formally introduce Mult-MSSC we present the results for the Online Version of Multistage Min-Sum Set Cover [28]. In this version we are given the request sets S_t one by one without knowing in advance what requests may come in the future. At each timestep t, the set S_t is revealed and we need to select a permutation π^t to serve this request. To move from permutation π^{t-1} to π^t our algorithm pays $d_{KT}(\pi^{t-1}, \pi^t)$ where d_{KT} denotes the Kendall-Tau distance. Our goal is to minimize $\sum_{t=1}^{T} \pi^t(S_t) + d_{KT}(\pi^{t-1}, \pi^t)$.

In [28] the authors investigate the Online Version of Min-Sum Set Cover. They notice that even though Online Min-Sum Set Cover is a generalization of List Update (List Update is a special case of Online Min-Sum Set Cover where all the request sets have cardinality 1, $|S_t| = 1$) the Move-to-Front which is the standard algorithm that achieves the optimal competitive ratio for List Update does not give any interesting results. In the future, as we proceed we consider that the request sets are r-uniform, that is $|S_t| = r$ for all $t \in [T]$.

1.3.1 A Memoryless Algorithm for Online Min-Sum Set Cover

The first result we present from the corresponding paper is a memoryless algorithm for Online Min-Sum Set Cover. An online algorithm is called memoryless iff the choices at timestep t are not dependent on previous timesteps. We present the Move-All-Equally(MAE) Algorithm. Essentially what we do is that when we receive a request set S_t we move every element of S_t in the current permutation π^{t-1} to the left until one of them reaches the head of the permutation. The algorithm is outlined below.

Algorithm 6 Move-All-Equally

```
Input: The given request at time t $S_t$ and the permutation $\pi^{t-1}$.
Output: A permutation at time $t$ $\pi^t$.
1: $k_t \leftarrow min\{i|\pi^{t-1}[i] \in S_t\}$
2: decrease the index of all elements on $S_t$ by $k_t - 1$.
```

This algorithm is quite interesting since it has 2 very interesting properties.

- Let e_t denote the element used by the optimal offline algorithm to cover the request S_t. Algorithm 6 moves this element towards the front of the list.
- It balances movement and access costs. If the access cost of the request S_t is k_t the movement cost of the algorithm is roughly $r \cdot k_t$. The basic idea is that the moving cost of algorithm 6 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.

Even though Algorithm 6 possesses these two interesting properties the authors show that it doesn't quite bridge the lower bound with a matching competitive ratio.

Lemma 5. *The Competitive Ratio of Algorithm 6 is* $\Omega(r^2)$.

Proof. To construct an adversarial input sequence for our lower bound we request every time the last r elements of the current permutation. Since Algorithm 6 moves all the elements at the same speed we construct n/r requests that are mutually disjoint. Therefore, we can serve each request optimally with a cost of $\Theta(n/r)$ whereas our MAE algorithm serves each request with a cost of $\Omega(n \cdot r)$. Therefore by comparing we obtain the lower bound on the competitive ratio. $\square$

Even though the analysis of this algorithm doesn't allow for a matching upper bound the authors provide us with the following result. For more details on the analysis of the algorithm the interested reader may refer to [28].

Lemma 6. *The Competitive Ratio of Algorithm 6 is at most* $O(2^{\sqrt{(logn \cdot logr)}})$.

1.3.2 The Lazy Rounding Algorithm

In this section we present a lower bound for every deterministic algorithm for Online Min-Sum Set Cover and further on we present the Lazy Rounding algorithm which is a deterministic algorithm that matches the lower bound (up to constant factors).

Theorem 5. *Any deterministic algorithm for the Online Min-Sum Set Cover problem has a competitive ratio at least* $(r+1)(1-\frac{1}{n+1})$.

For proving this Theorem the authors use an averaging argument, similar to that of List-Update [40]. In each step the adversary requests once again the last r elements of the current permutation and therefore the access cost is at least (n-r+1). By using simple counting techniques they show that for any fixed set S_t of size r and any $i \in [n-r+1]$, the number of permutations π with access cost $\pi(S_t) = i$ is $\binom{n-i}{r-1} \cdot r! \cdot (n-r)!$. By summing over all permutations and dividing by $n!$ to get the average cost we get that the optimal permutation has cost at least $\frac{n+1}{r+1}$ and the competitive ratio of any algorithm is at lteat $(r+1)(1-\frac{1}{n+1})$. For more details the interested reader may refer to the original paper.

Before stating the Lazy Rounding Algorithm and the main results we mention some of the algorithms that may sound natural but behave trivially bad. What's interesting is that this behaviour translates directly even in the offline version of the problem so it's interesting to know what absolutely doesn't work and motivate a non-combinatorial approach for the Mult-MSSC.

- MTF_{first}: Move to the front of the permutation the element of S_t that appears first in π^t.
- MTF_{last}: Move to the front of the permutation the element of S_t that appears last in π^t.
- MTF_{random}: Move a random element of S_t to the front of the list.

All these algorithms have a tight $O(n)$ competitive ratio. Below we highlight the main parts of the Lazy Rounding algorithm which has an $O(r)$ competitive ratio.

1. Use a black-box Multiplicative Weights Update Algorithm (MWU) with learning rate $1/n^3$. Using the results from learning theory the authors show that: $AccessCost(MWU) \le \frac{5}{4} AccessCost(OPT)$.
2. Using an online rounding scheme which turns any randomized algorithm A to a deterministic one, denoted $Derand(A)$, with access cost at most $2r \cdot \mathbb{R}[AccessCost(A)]$, without any guarantee so far for the Moving Cost of $Derand(A)$.
3. Lazy-Rounding is essentially a lazy version of $Derand(MWU)$ (applying the online rounding scheme to the Multiplicative Weights Update Algorithm) that updates the permutation only if the distribution of MWU has changed a lot. Let's call a time interval where Lazy-Rounding does not change it's permutation a phase. The authors show that during a phase:

- $AccessCost(Lazy - Rounding) \leq 4r \cdot \mathbb{E}[AccessCost(MWU)]$.
- $MovingCost(Lazy - Rounding) \leq \mathbb{E}[AccessCost(MWU)]$.

By combining the above properties we obtain:

$$Cost(Lazy - Rounding) \leq (4r+1)\mathbb{E}[AccessCost(MWU)] \leq (5r+2)Cost(OPT)$$

.

Theorem 6. *The deterministic online algorithm Lazy-Rounding is (5r+2) competitive for the Online Version of Min-Sum Set Cover.*

1.4 The Multistage Min-Sum Set Cover

1.4.1 Problem Definition

In *Multistage Min-Sum Set Cover* (Mult-MSSC), we are given a universe U on n elements, a sequence of requests $\mathcal{R} = (R_1, \dots, R_T)$, with $R_t \subseteq U$, and an initial permutation π^0 of the elements of U. We aim to maintain a sequence of permutations $(\pi^0, \pi^1, \dots, \pi^T)$ of U, so as to minimize the total cost of updating (or moving from) π^{t-1} to π^t in each time step plus the total cost of covering each request R_t with the current permutation π^t. The cost of moving from π^{t-1} to π^t is the number of inverted element pairs between π^{t-1} and π^t, i.e., the Kendall Tau distance $\mathrm{d_{KT}}(\pi^{t-1}, \pi^t)$. The cost $\pi^t(R_t)$ of covering a request R_t with a permutation π^t is the position of the first element of R_t in π^t, i.e., $\pi^t(R_t) = \min\{i \,|\, \pi^t(i) \in R_t\}$. Thus, given $\mathcal{R} = (R_1, \dots, R_T)$, we aim to minimize $\sum_{t=1}^{T} \left(\mathrm{d_{KT}}(\pi^{t-1}, \pi^t) + \pi^t(R_t)\right)$.

The Mult-MSSC problem is a natural generalization of the (offline version of the) classical *List Update* problem [41], where $|R_t| = 1$ for all requests $R_t \in \mathcal{R}$. The offline version of List Update is NP-hard [4], while it is known that any $5/4$-approximation has to resort to *paid exchanges*, where an element different from the requested one is moved forward to the list [37, 43]. Mult-MSSC was introduced in [28] as the multistage extension of Min-Sum Set Cover (MSSC) [24], where we aim to compute a single static permutation π that minimizes the total covering cost $\sum_{t=1}^{T} \pi(R_t)$. [28] presented a (simple polynomial-time) online algorithm for Mult-MSSC with competitive ratio between $\Omega(r\sqrt{n})$ and $O(r^{3/2}\sqrt{n})$ for r-bounded instances, where all requests have cardinality at most r, and posed the polynomial-time approximability of Mult-MSSC as an interesting open question. Mult-MSSC is also related to recently studied time-evolving (a.k.a. multistage or dynamic) optimization problems (e.g., multistage matroid, spanning set and perfect matching maintenance [30], time-evolving Facility Location [21, 5]), where we aim to maintain a sequence of near-optimal feasible solutions to a combinatorial optimization problem, in response to time-evolving underlying costs, without changing too much the solution from one step to the next.

1.4.2 Motivation

Mult-MSSC is motivated by applications, such as web search, news, online shopping, paper bidding, etc., where items are presented to the users sequentially. Then, the item ranking is of paramount importance, because user attention is usually restricted to the first few items in the sequence (see e.g., [42, 20, 26, 12]). If a user does not spot an item fitting her interests there, she either leaves the service (in case of news or online shopping, see e.g., the empirical evidence in [18]) or settles on a suboptimal action (in case of paper bidding, see e.g., [15]). To mitigate such situations and increase user retention, modern online services highly optimize item rankings based on user scrolling and click patterns. Each user t is represented by her set of preferred items (or item categories) R_t. The goal of the service provider is to continually maintain an item ranking π^t, so that the current user t finds one of her favorite items at a relatively high position in π^t. Continual ranking update is dictated by the fact that users with different characteristics and preferences tend to use the online service during the course of the day (e.g., elderly people in the morning, middle-aged people in the evening, young people at the night – similar patterns apply for people from different countries and timezones). Moreover, different user categories react in nonuniform ways to different trends (in e.g., news, fashion, sports, scientific topics). For consistency and stability, however, the ranking should change neither too much nor too frequently. Mult-MSSC makes the (somewhat simplifying) assumptions that the service provider has a relatively accurate knowledge of user preferences and their arrival order, and that its total cost is proportional to how deep in π^t the current user t should reach, before she finds one of her favorite items, and to how much the ranking changes from one user to the next.

From a theoretical viewpoint, Mult-MSSC was used in [28] as a natural benchmark for studying the dynamic competitive ratio of Online Min-Sum Set Cover, where the algorithm updates its permutation online, without any knowledge of future requests. As in Mult-MSSC, the objective is to minimize the total moving plus the total covering cost.

1.4.3 Our Contribution and Techniques.

In this work, we initiate a study of the polynomial-time approximability of Mult-MSSC. Using a reduction from Set Cover, we show (Theorem 8) that Mult-MSSC does not admit a $c \log n$-approximation, for some absolute constant c, unless P $=$ NP. Moreover our reduction establishes that an $o(r)$-approximation for r-bounded instances of Mult-MSSC implies an $o(r)$-approximation for Set Cover, in case each element appears in at most r requests.

Our main technical contribution is to show that Mult-MSSC can be approximated in polynomial-time within a factor of $O(\log^2 n)$ in general instances, by randomized rounding (Theorem 9), and within a factor of $O(r^2)$ in r-bounded instances, by deterministic rounding (Theorem 10).

For both results, we consider a restricted version of Mult-MSSC, inspired by the Move-to-Front (MTF) algorithm for List Update, where in each time step t, we can only move a single element of R_t from its position in π^{t-1} to the first position of π^t. Since such a permutation π^t coves R_t with unit cost, we now aim to select the element of each R_t moved to front of π^t, so as to minimize the total moving cost $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^t)$. It is not hard to see that the optimal cost of serving $\mathcal{R}$ under the restricted Move-to-Front version of Mult-MSSC is within a factor of 4 from the optimal cost under the original, more general, definition of Mult-MSSC.

Hence, approximating Mult-MSSC boils down to determining which element of R_t should become the top element of π^t. To this end, we relax permutations to doubly stochastic matrices and consider a Linear Programming relaxation of the restricted Move-to-Front version of Mult-MSSC, which we call *Fractional-MTF* (see Definition 12). Given the optimal solution of the aforementioned linear program, which is a sequence of doubly stochastic matrices $(A^0, A^1, \dots, A^T)$, with A^0 corresponding to the initial permutation π^0, our main technical challenge is to round each doubly stochastic matrix A^t to a permutation π^t such that (i) there is an element of R_t at one of the few top positions of π^t; and (ii) the total moving cost $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, \pi^t)$ of the rounded solution is comparable to the total moving cost $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^{t-1}, A^t)$ of the optimal solution of Fractional-MTF, where d_{FR} is a notion of distance equivalent to Spearman's footrule distance on permutations (see Definition 10).

Working towards a randomized rounding approach, we first observe that rounding each doubly stochastic matrix independently may result in a permutation sequence with total moving cost significantly larger than that of Fractional-MTF (see also the discussion after Lemma 12). In Theorem 9, we show that a dependent randomized rounding with logarithmic scaling of entries (Algorithm 8), similar in spirit with the randomized rounding approach [10, 39] for Generalized Min-Sum Set Cover, results in an approximation ratio of $O(\log^2 n)$. Interestingly, Algorithm 8 without the logarithmic scaling results in a permutation sequence with the expected moving cost within a factor of 4 from the optimal moving cost of Fractional-MTF. However, we lose a logarithmic factor in the approximation ratio, because we need to scale up the entries of each doubly stochastic matrix A^t, so as to ensure that some element of R_t appears in the few top positions of π^t with sufficiently large probability. The other logarithmic factor is lost because there could be a logarithmic number of elements allocated to the same position of the resulting permutation by the randomized rounding.

Our deterministic rounding of Algorithm 9 for r-bounded request sequences is motivated by the deterministic rounding for Set Cover and Vertex Cover. We observe that in the optimal solution of Fractional-MTF, in each time step t, there is some element $e \in R_t$ with $A^t_{e1} \geq 1/r$ (i.e., e occupies a fraction of at least $1/r$ of the first position in the "fractional permutation" A^t). Algorithm 9 simply moves any such element to the front of π^t. The most challenging part of the analysis is to establish that for any optimal solution $(A^0, A^1, \ldots, A^T)$ of Fractional-MTF with respect to an r-bounded request sequence, there exists a sequence of doubly stochastic matrices $(A^0, \hat{A}^1, \ldots, \hat{A}^T)$ with the entries of each $\hat{A}^t$ being multiples of $1/r$, such that (i) the moving cost of $(A^0, \hat{A}^1, \ldots, \hat{A}^T)$ is bounded from above by the optimal cost of Fractional-MTF; and (ii) each matrix $\hat{A}^t$ contains in the first position the element that Algorithm 9 keeps in the first position at round t, with mass at least $1/r$. Then we show (Lemma 16) that for any sequence of doubly stochastic matrices $(A^0, \hat{A}^1, \ldots, \hat{A}^T)$ satisfying the above properties, the moving cost of Algorithm 9 is at most the moving cost of the doubly stochastic matrices, $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1})$. The latter is done through the use of an appropriate potential function based on an extension of the Kendall-Tau distance to doubly stochastic matrix with entries being multiples of $1/r$.

A potentially interesting insight is that the technical reason for the quadratic dependence of our approximation ratios on $\log n$ and r is conceptually similar to the reason for the (best possible) approximation ratio of $4 = 2 \cdot 2$ in [24] (see the discussion after Theorem 9). Hence, we conjecture that any $o(\log^2 n)$ (resp. $o(r^2)$) approximation to Mult-MSSC must imply a sublogarithmic (resp. $o(r)$) approximation to Set Cover.

Chapter 2

Previous Work and Related Problems

In this chapter we study a general class of problems that are closely related to our problem of interest, that is Mult-MSSC. As we've already examined in the introduction, Mult-MSSC is a problem which evolves over time and our aim is to hold a sequence of states (in our case permutations) such that we optimize an objective function. It is fairly common to consider the time evolving case of a well known problem. Usually we need a notion of a moving cost so that the decisions we make affect the trajectory of the objective function in the future. One such problem is considered in section 2.2 where we present part of the work at [27].

Facility Location is a classical problem that has been widely studied in both combinatorial optimization and operations research, due to its many practical applications. It provides a simple and natural model for industrial planning, network design, machine learning, data clustering and computer vision [19, 36, 16, 13]. In its most basic form, the input to a Facility Location setting is a metric space and the location of n *agents*. The designer then selects another point in the space to serve as the *facility*, with the objective being to minimize the sum of distances to the agents. The K-Facility Location problem, which is also widely known as the K-median problem, is a straightforward generalization where K facilities are used instead. The problem then becomes more reminiscent of *clustering*, with the added subtlety of also having to match facilities with agents.

Many times in optimization problems that are time-evolving, it is natural to assume that we have no information regarding how the data will evolve over time. If we consider real world applications this is almost always the case. This kind of setting is called online and we aim to design online algorithms that perform not a lot worse than the corresponding optimal offline algorithm (that is, we compare the performance of our algorithm to the algorithm that knows everything in advance). This idea is captured in the notion of competitiveness defined below.

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

$$ALG(\sigma) \leq c \cdot OPT(\sigma) + b$$

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

To demonstrate basic techniques from the field of online optimization we introduce the List Update Problem. Mult-MSSC generalizes List Update for requests of cardinality greater than 1 but several techniques that we use in analysing our algorithms further on such as the Potential Function or the Move-To-Front algorithm are nicely outlined in the List Update Problem. For a more thorough review of the problem as well as other online problems the interested reader may refer to [1, 2, 43, 40].

2.1 The List Update Problem

The List Update problem is one of the classical and well studied problems in online optimization [40]. We are given a permutation of $\{1, 2, ..., n\}$ and set of request elements $r_1, r_2, ..., r_m$ where $r_i \in \{1, 2, ..., n\}$. For each request r_i we pay the cost of accessing it which is the position of the element r_i in the current permutation. Then, we can choose to move this element closer to the front of the list for free in order to accommodate future requests.

In the online version of the problem when we are given a specific request we don't know anything about future requests.

2.1.1 A naive algorithm for List Update

A natural first attempt for an algorithm is to order the list based on the current access frequency of the keys. Intuitively, this algorithm seems reasonable since if we want to keep our overall search cost low, we want more frequently requested elements to be closer to the front of the list. Also note this ordering can be maintained in the model of the problem, since when the currently requested key can only move up infrequency rank when it is requested.Unfortunately, the following lemma shows that ordering by frequency is trivially bad with respect to competitive analysis.

Lemma 7. *The* Order By Frequency *algorithm is $\Omega(n)$ competitive.*

Proof. Suppose we start from the sorted permutation $[1, 2, 3, ..., n]$. Consider the following request sequence: element 1 requested n times, element 2 requested n times, ..., element n requested n times. Now consider the frequency cost of the second half elements requested, that is elements $n/2, n/2 + 1, ..., n$. Each of these requests has a cost of at least $n/2$ since the first $n/2$ keys all have a cost of n. Therefore, the total cost of the algorithm is $\Omega(n^3)$.

Next, we show that there is an algorithm that achieves $O(n^2)$ total cost. If we move each requested element to the front of the list we can see that the overall total cost is $O(n^2)$. Therefore, the competitive ration of the order by frequency algorithm is $\Omega(n)$. □

2.1.2 The Move to Front Algorithm

Motivated by the proof of lemma 7 we propose a natural algorithm for the List Update Problem. Move the requested element to the front of the list. This algorithmic schema that is also used on the next chapter although simple achieves $O(1)$ competitiveness. In particular the Move to Front(MTF) algorithm is 2-competitive for the List Update problem.

Lemma 8. *The MTF Algorithm is 2-Competitive for List Update.*

2.1.3 Proof of Lemma 8

To prove the 2-competitiveness of the MTF algorithm we use the potential method. We define a function $\Phi : [0, m] \to \mathbb{Z}$ that maps the state of the permutation after each request to an integer. Our aim is for $\Phi(i)$ to capture the distance of our permutation from the optimal permutation after serving the request r_i.

Next, we define the amortized cost based on our potential function. Let $A(i) = MTF(i) + \Phi(i) - \Phi(i-1)$ where $MTF(i)$ is the cost that Move-To-Front pays for serving the i-th request. Observe that:

$$\sum_{i=1}^{m} A(i) = \sum_{i=1}^{m} MTF(i) + \Phi(i) - \Phi(i-1) = \Phi(m) - \Phi(0) + \sum_{i=1}^{m} MTF(i) \qquad (2.1)$$

Therefore, if we define the potential function in way such that $\Phi(m) - \Phi(0) > 0$ then the sum of the amortized cost for all the requests is an upper bound for the cost of our MTF algorithm.

To define our potential function, let $V(i)$ be the set of pairs (x, y) such that x in the permutation maintained by MTF is inverted with respect to y in the optimal permutation after serving the request i. To clarify what this means, let $Pos(\pi^i, x)$ be the index of x in the permutation π^i, that is $Pos(x) = (\pi^i)^{-1}(x)$. Let o^i be the optimal permutation at time i, that is the permutation maintained by the optimal offline algorithm after serving the request i. Now we define:

$$V(i) = \{(x, y) \in [1, n] \times [1, n] \text{ s.t } Pos(\pi^i, x) < Pos(\pi^i, y) \wedge Pos(o^i, x) > Pos(o^i, y)$$

So $V(i)$ is the set of pairs (x,y) such that x comes before y in the permutation maintained by MTF but x comes after y in the optimal permutation. We define $\Phi(i) = |V(i)|$ and now we are ready to state our lemma:

Claim 1. *Let $A(i)$ be the amortized cost as defined on 2.1. Let $OPT(i)$ be the cost of the optimal solution after serving the i-th request, then $A(i) \leq 2 \cdot OPT(i) - 1$.*

Proof. Let r_i be the i-th request. Let k be the number of elements that come before r_i in both the MTF and the optimal permutation and let l be the number of elements that come before r_i in the MTF permutation but after r_i in the optimal permutation. Based on these definitions we have $MTF(i) = l + k + 1$ and also $OPT(i) \geq k + 1$. Now observe that if we move r_i to the front of the list then we create at most k new inversions and we destroy exactly l inversions. Therefore $\Phi(i) - \Phi(i-1) \leq k - l$. Putting everything together we obtain:

$$A(i) = MTF(i) + \Phi(i) - \Phi(i-1) \leq l + k + 1 + (k - l) = 2 \cdot k + 1 \leq 2 \cdot (OPT(i) - 1) \leq 2 \cdot OPT(i) - 1$$

Which finishes the proof. □

Now we are ready to prove that MTF is 2-competitive for List Update. By combining 2.1 and lemma 1 we obtain $MTF = \sum_{i=1}^{m} MTF(i) \leq 2 * OPT - m + \Phi(0) - \Phi(m) \leq 2 * OPT - m + \binom{n}{2}$. The final step is obtained from the fact that $\Phi(0) = 0$ and $\Phi(i) \leq \binom{n}{2}$ since the number of inversions are always bounded by $\binom{n}{2}$.

2.1.4 Lower Bound on the Competitive Ratio

In this section we prove that any online algorithm for the List Update problem is at least 2-competitive.

Proof. For any online algorithm produce an adversarial instance as follows: Set request r_i equal to the last element of the current permutation. Obviously, for a permutation of length n and m requests the cost of any online algorithm is nm. We will show that there is a sequence of permutations such that its total cost is at most $\frac{nm}{2}$.

Consider a sequence of random permutations. The expected cost for serving the request r_i is $\frac{n}{2}$ therefore the expected cost of the random sequence of permutations is $\frac{nm}{2}$. By a probabilistic argument we obtain that there is a sequence of permutations with cost at most $\frac{nm}{2}$ which finishes the proof. □

2.2 Reallocating Multiple Facilities on the Line

In this section, we start by presenting the K-Facility Reallocation problem [27] via an example and then we move to the formal definitions of the concepts, which appear frequently throughout the section.

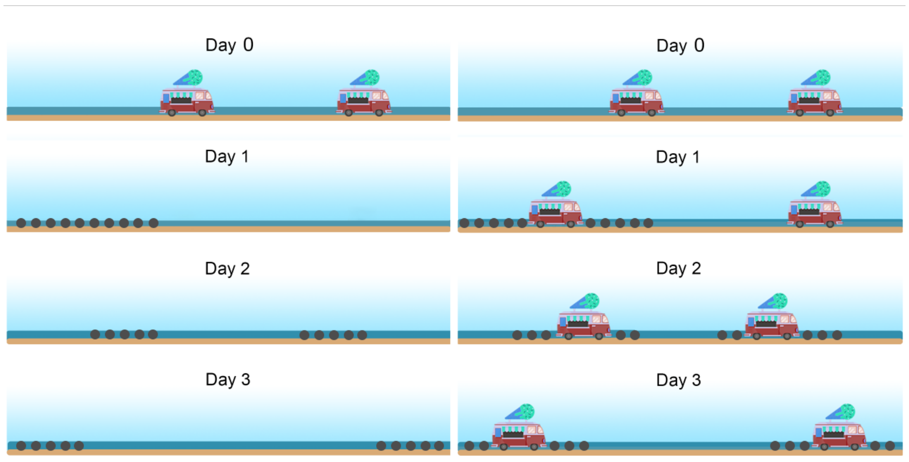


Figure 2.1: Depiction of the simple example of Section 2.2.1. The picture on the left represents an instance of the 2-Facility Reallocation problem on the line. The initial ice cream vendor locations are depicted in day 0 and the dots indicate the customer locations. On the right picture, we illustrate a good solution for this instance, namely the facility locations at each day.

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 plus the total distance traveled by the ice cream vendors. Figure 2.1 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 problem is 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. The following definitions formalize these concepts.

The underlying assumption so far has usually been that the agent locations are known in advance, but this may not be the case in general. For example, in many natural settings such as network design or providing targeted content given preference clusters, the ‘agent locations’ are not known in advance. Motivated by this fact, Meyerson [38] introduced online facility location problems, where agents arrive one-by-one and must be *immediately and irrevocably* assigned to a facility upon arrival. Moreover, the set of agent locations is revealed in either random or adversarial order. However, all connections are irrevocable and cannot be changed even if new facilities are added, reflecting the constraints of adding physical links in a network.

More recently, understanding the dynamics of temporally evolving social or infrastructure networks has been the central question in many applied areas such as viral marketing, urban planning etc. Dynamic Facility Location proposed in similar versions by [22] and [25] has been a new tool to analyze the temporal aspects of such networks. In this time dependent variant of facility location, the *underlying metric space* can change over time. This can be perceived as agents moving through space, but is actually more flexible. Facilities cannot be moved around however: amongst the set of available facilities, the algorithm can select a subset to open at a cost. Then, at each turn, the assignment of agents to facilities can be changed. However, this incurs a *switching cost* and the objective is to achieve the best tradeoff between the optimal connections of agents to facilities and the stability of solutions between consecutive timesteps. This switching cost can be interpreted differently depending on the situation modelled, but taking it into account can produce significantly different (and more 'reasonable' looking) clusters than statically finding the best assignment each turn.

While the previous models have addressed issues of dynamically maintaining a set of facilities (or clusters), the common theme has been that facilities cannot move. They can be dissolved and reinstated somewhere else, but the cost is usually measured in terms of active facilities at the end. Our contribution is to study such settings where facilities can also *move* around.

2.2.1 Problem Definition and Preliminaries

Definition 2 (K*-Facility Reallocation problem*)**.** *We are given a tuple* (x^0, C) *as input. The* K *dimensional vector* $x^0 = (x_1^0, \dots, x_K^0)$ *describes the initial positions of the facilities. The positions of the agents over time are described by* $C = (C_1, \dots, C_T)$*. The position of agent* i *at stage* t *is* α_i^t *and* $C_t = (\alpha_1^t, \dots, \alpha_n^t)$ *describes the positions of the agents at stage* t*.*

Definition 3. *A solution to the K-Facility Reallocation problem is a sequence* $x = (x^1, \dots, x^T)$*. Each* $x^t = (x_1^t, \dots, x_K^t)$ *is a* K *dimensional vector that gives the positions of the facilities at stage* t *and* x_k^t *is the position of facility* k *at stage* t*. The cost of the solution* x *is*

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

Given an instance (x^0, C) of the problem, the goal is to find a solution x that minimizes the $\text{Cost}(x)$. The term $\sum_{t=1}^{T} \sum_{k=1}^{K} |x_k^t - x_k^{t-1}|$ describes the cost for moving the facilities from place to place in a solution x. We refer to it as *moving cost* and is denoted as $\text{MovCost}(x)$. The term $\sum_{t=1}^{T} \sum_{i=1}^{n} \min_{1 \le k \le K} |\alpha_i^t - x_k^t|$ describes the connection cost of the agents in a solution x. We refer to it as *connection cost* and is denoted as $\text{ConCost}(x)$. The connection cost of an agent i at stage t in solution x is denoted as $\text{ConCost}_i^t(x)$ and is the quantity $\min_{1 \le k \le K} |\alpha_i^t - x_k^t|$.

Observe that the competitive ratio of an online algorithm involves an additive constant b in its definition. The additive constant may depend on the initial configuration of the problem (in our case the initial positions of the facilities) but not on the request sequence σ. This is the standard definition used in K-server like problems (see also [35, 14]).

2.2.2 A Polynomial Time Algorithm for K-Facility Reallocation Problem

In this section, we present an algorithm which computes an optimal solution for the K-Facility Reallocation problem on the line with running time polynomial in n, T and K. Our approach is a typical LP based algorithm that consists of two basic steps.

- **Step 1:** Expressing the K-Facility Reallocation problem as an Integer Program.
- **Step 2:** Solving *fractionally* the relaxation of the Integer Program and *rounding* the fractional solution to an integral one.

The main technical contribution is showing that a simple rounding scheme yields an integral solution that has the exact same cost with an optimal fractional solution.

2.2.3 Presentation of the offline algorithm

We start by expressing the K-Facility Reallocation problem on the line as an Integer Program. The following lemma from [17] suggests that we can focus on finite locations on the real line.

Fact 1 (Lemma 2 in [17])**.** *Let (x_0, C) be an instance of the K-Facility Reallocation problem. There exists an optimal solution x^* such that for all stages $t \in [T]$ and $k \in \{1, \dots, K\}$,*

$$x_k^{*t} \in C_1 \cup \ldots \cup C_T \cup x^0.$$

According to Fact 1, there exists an optimal solution that places the facilities only at locations where either an agent has appeared or a facility was initially lying. This allows us to focus only on solutions which place facilities on at most $K + Tn$ different locations on the real line. Fact 1 provides an exhaustive search algorithm for the problem and is also the basis for the *Dynamic Programming* approach in [17], which has running time $O(T^2(\max\{Tn, k\})^{k+1})$. We will use Fact 1 to formulate our Integer Program.

Specifically, let Loc $:= C_1 \cup \ldots \cup C_T \cup x^0$ denote the set of the locations of Fact 1. This set can be equivalently represented by a *locations path* $P = (V, E)$. That is, if we sort the locations of Loc in non decreasing order, then the jth location of Loc is the jth node of the locations path P and the distance between two nodes in P is the distance of the corresponding locations on the line. Now, the K-Facility Reallocation problem on the line takes the following *discretized form*: For an instance (x^0, C), we can construct a locations path $P = (V, E)$. Each facility is initially located at a node $j \in V$ and at each stage t, each agent is also located at a node of P. The goal is to move the facilities from node to node such that the connection cost of the agents plus the moving cost of the facilities is minimized.

To formulate this discretized version as an Integer Program, we introduce some additional notation. Let $\mathrm{d}(j, l)$ be the distance of the nodes $j, l \in V$ in P and A be the set of agents. For each $i \in A$, $\mathrm{Loc}(i, t)$ is the node where agent i is located at stage t. We also define the following $\{0, 1\}$-indicator variables for all $t \in [T]$: $\zeta_{ij}^t = 1$ if, at stage t, agent i connects to a facility located at node j, $f_{kj}^t = 1$ if, at stage t, facility k is located at node j, $S_{kjl}^t = 1$ if facility k was at node j at stage $t - 1$ and moved to node l at stage t. The variable c_j^t represents the number of facilities that are placed on node j at stage t. This problem can be formulated as the Integer Program depicted in Figure 2.2.

Constraints (1)-(4) of the Integer Program are the standard K-median constraints with the only difference that the facilities are distinguished, each having a different label. Facility m is the mth leftmost facility according to the initial facility locations ordered from left to right. Constraints (5)-(7) describe a flow problem and are used to model the moving cost incurred by the movement of facilities between stages. Specifically, constraints (1)-(3) correspond to the fact that at every stage t, each agent i must be connected to a node j where at least one facility k is located. The constraint $\sum_{j \in V} f_{kj}^t = 1$ enforces each facility k to be located at exactly one node j. The constraint $S_k^t = \sum_{j,l \in V} \mathrm{d}(j, l) S_{kjl}^t$ describes the cost for moving facility k from node j to node l. Constraints (6)-(7) ensure that facility k moved from node j to node l at stage t if and only if facility k was at node j at stage $t - 1$ and was at node l at stage t ($S_{kjl}^t = 1$ iff $f_{kl}^t = 1$ and $f_{kj}^{t-1} = 1$). Observe that the values of f_{kj}^0 are determined by the initial positions of the facilities, which are given by the instance of the problem.

For ease of exposition, we will refer to the Integer Program of the K-Facility Reallocation problem as $\mathrm{IP}_{\mathrm{Rllctn}}$ and we will refer to the relaxation of the $\mathrm{IP}_{\mathrm{Rllctn}}$ as $\mathrm{LP}_{\mathrm{Rllctn}}$. An optimal fractional solution of the $\mathrm{LP}_{\mathrm{Rllctn}}$ is denoted as Z_{LP}.

Our algorithm, described in Algorithm 7, is a simple rounding scheme of an optimal fractional solution Z_{LP} for the $\mathrm{LP}_{\mathrm{Rllctn}}$. At each stage t, the rounding procedure starts from the first node from the left and finds for each facility m, $1 \leq m \leq K$, the sum of the total facility amount placed on the visited nodes until a node j_m^t makes this sum greater than $m - 1$. Then, the algorithm places facility m to the location on the line, which corresponds to node j_m^t. Observe, that this procedure leads to a placement of the facilities on the leftmost position of their fractional appearance ($f_{kj} > 0$) in Z_{LP} at each stage.

$$\text{minimize} \sum_{t=1}^{T} \left[\sum_{i \in C} \sum_{j \in V} \mathrm{d}(\mathrm{Loc}(i,t), j)\zeta_{ij}^t + \sum_{k \in F} S_k^t \right]$$

$$\begin{array}{llr}
(1) & \sum_{j \in V} \zeta_{ij}^t = 1 & \forall i \in A, t \in [T] \\
(2) & \zeta_{ij}^t \le c_j^t & \forall i \in A, j \in V, t \in [T] \\
(3) & c_j^t = \sum_{k=1}^{K} f_{kj}^t & \forall j \in V, t \in [T] \\
(4) & \sum_{j \in V} f_{kj}^t = 1 & \forall k \in [K], t \in [T] \\
(5) & S_k^t = \sum_{j,l \in V} \mathrm{d}(j,l) S_{kjl}^t & \forall k \in [K], t \in [T] \\
(6) & \sum_{j \in V} S_{kjl}^t = f_{kl}^t & \forall l \in V, k \in [K], t \in [T] \\
(7) & \sum_{l \in V} S_{kjl}^t = f_{kj}^{t-1} & \forall j \in V, k \in [K], t \in [T] \\
(8) & \zeta_{ij}^t, f_{kj}^t, S_{klj}^t \in \{0,1\} & \forall j \in V, k \in [K], t \in [T]
\end{array}$$

Figure 2.2: Integer Program formulation of the K-Facility Reallocation problem

Algorithm 7 Algorithm for the offline case

```
Input: The initial positions $x^0 = \{x_1^0, \ldots, x_K^0\}$ of the facilities and the positions of the agents $C = \{C_1, \ldots, C_T\}$.
  Construct the path $P$ and the IP$_{\mathrm{Rllctn}}$ (Figure 2.2).
  Solve the relaxation LP$_{\mathrm{Rllctn}}$ of the IP$_{\mathrm{Rllctn}}$.
  for each stage $t \ge 1$: do
    for $m = 1, \ldots, K$, find the node $j_m^t$ such that $\sum_{\ell=1}^{j_m^t - 1} c_\ell^t \le m - 1 < \sum_{\ell=1}^{j_m^t} c_\ell^t$ do
      Locate facility $m$ at the respective location $x_m^t$ the line, namely $x_m^t \leftarrow \mathrm{d}(j_m^t, 1) + \min_{p \in C_1 \cup \ldots \cup C_T \cup x^0} p$.
    end for
  end for
```

Our goal is to show that the aforementioned rounding scheme produces an integral solution that has the exact same cost with $\mathrm{Z_{LP}}$. The next section is dedicated to highlight the core ideas and basic steps of this fact.

2.2.4 Overview of our approach

In this section, we discuss the core ideas and basic steps for proving that Algorithm 7 provides an optimal solution for the K-Facility Reallocation problem on the line. Our goal is to show the following theorem.

Theorem 7. *For the* LP$_{\mathrm{Rllctn}}$*, there exist* N *integral solutions* Sol_p*,* $1 \le p \le N$*, for some integer* $N > 0$*, such that:*

- $\frac{1}{N}\sum_{p=1}^{N} \text{ConCost}_i^t(\text{Sol}_p) = \sum_{j\in V} \text{d}(\text{Loc}(i,t),j)\zeta_{ij}^t$ *(1)*
- $\frac{1}{N}\sum_{p=1}^{N} \text{MovCost}(\text{Sol}_p) = \text{MovCost}(\text{Z}_{\text{LP}})$ *(2)*

Furthermore, the solution produced by Algorithm 7 is one of the N integral solutions Sol_p*,* $1 \le p \le N$.

Theorem 7 directly implies that the total cost of each integral solution Sol_p equals the cost of the optimal fractional solution Z_{LP}. Unfortunately, the number N of the integral solutions Sol_p can be exponential with respect to the parameters of the problem, thus producing all of them is not an option. However, we will show that the rounding scheme of Algorithm 1, which clearly runs in polynomial time with respect to n, K and T, produces one of these solutions, namely Sol_1.

The fact that each integral solution Sol_p, $1 \le p \le N$, has the same cost with an optimal fractional solution follows easily from Theorem 7. Observe that by taking a sum over all clients and all stages in the first equation of Theorem 7, we have that the average connection cost of the solutions Sol_p equals the connection cost of Z_{LP}. Combining this result with the second equation of Theorem 7, we have that the solutions Sol_p have average cost equal to the optimal fractional cost, i.e., $\frac{1}{N}\sum_{p=1}^{N}\text{Cost}(\text{Sol}_p) = \text{Cost}(\text{Z}_{\text{LP}})$. Since $\text{Cost}(\text{Sol}_p) \ge \text{Cost}(\text{Z}_{\text{LP}})$ for every p, $1 \le p \le N$, it must hold that $\text{Cost}(\text{Sol}_p) = \text{Cost}(\text{Z}_{\text{LP}})$ for every p, $1 \le p \le N$.

Due to technical and intuitive reasons, we break down the proof of Theorem 7 into two steps. In the first step, which we exhibit in Section 2.2.5, we prove that Theorem 7 holds if we further assume that the values of the variables in an optimal fractional solution are either $1/N$ or 0. Formally,

Assumption 1. *There exists a positive integer N, such that the variables f_{kj}^t and c_j^t of the* $\text{LP}_{\text{Rllctn}}$ *have value either $1/N$ or 0.*

We use Assumption 1 to define the integral solutions Sol_p that are the main component of our proofs. Each solution Sol_p places facilities on a subset of nodes with positive c_j^t.

Definition 4. *V_t^+ denotes the set of nodes of the locations path P with a positive amount of facility (c_j^t) at stage t,*

$$j \in V_t^+ \Leftrightarrow c_j^t = 1/N\,.$$

We remind that since c_j^t is either $1/N$ or 0 then $|V_t^+| = K \cdot N$. We also consider the nodes in $V_t^+ = \{Y_1^t, \dots, Y_{K\cdot N}^t\}$ to be ordered from left to right. Now, we can formalize the concept of the solutions Sol_p, which will be used throughout the proofs. The term mth facility in the following definition of the integral solutions Sol_p refers to the ordering of the facilities on the real line according to their initial locations $\{x_1^0, \dots, x_K^0\}$.

Definition 5. Sol_p *denotes the integral solution that at each stage t places the mth facility at the $(m-1)N + p$ node of V_t^+ ,i.e., $Y_{(m-1)N+p}^t$, where $1 \le p \le N$.*

Since the solutions Sol_p play an important role in the understanding of the proof details, we provide an example of their structure in Figure 2.3.

Observation 1. Sol_1*, which is the integral solution that at each stage t places the mth facility at node $Y_{(m-1)N+1}^t$, is the solution produced by Algorithm 7.*

Since Algorithm 7 provides one of the solutions Sol_p, it remains to show the two claims of Theorem 7. We start by demonstrating the high-level idea for the proof of the first claim.

The core idea of the proof is that agent i connects to N nodes in Z_{LP}, on which the N distinct solutions Sol_p have placed a facility, at stage t. Specifically, i finds the N closest nodes N_i^t of V_t^+ in Z_{LP} and receives $1/N$ amount of service from each facility on these nodes. The nodes of N_i^t are consecutive nodes of V_t^+. By the construction of the solutions Sol_p, there exists a unique node in N_i^t where each Sol_p places a facility. Thus, the average connection cost of i in the integral solutions is the same with the connection cost of i in Z_{LP} at stage t. We illustrate these ideas Figure 2.4 and present the formal proof in Lemma 9.

Figure 2.3: The figure above is a graphical illustration of the optimal fractional solution Z_{LP} (top) together with the integral solutions Sol_p (bottom). For this instance $N = 3, T = 4$ and $K = 2$. Z_{LP} puts $1/N = 1/3$ amount of facility on some nodes at each stage. These nodes are the nodes of V_t^+, $1 \leq t \leq T$, which are grouped in $K = 2$ blocks with $N = 3$ nodes each ($|V_t^+| = 6$). Starting from the left, Sol_1 (asterisk) puts the two facilities on the first node of each block of V_t^+, Sol_2 (triangle) puts the two facilities on the second node of each block of V_t^+ and Sol_3 (cross) puts the two facilities on the third node of each block of V_t^+, at each stage t. Sol_1 is the solution, which Algorithm 7 produces for this instance.

Figure 2.4: In this example, we illustrate the relation of the integral solutions Sol_p with the optimal fractional solution Z_{LP} in terms of connecting agents. We have 2 facilities ($K = 2$), $N = 3$ and an agent i on the 5th node (with black color) from the left. The figure on the top depicts the placement of the $\zeta_{ij}^t = 1/N = 1/3$ amount in Z_{LP}, where i receives $1/3$ amount of service from his N closest facilities. The figure on the bottom shows the connection of the same agent in the integral solutions $\mathrm{Sol}_1, \mathrm{Sol}_2$ and Sol_3.

Figure 2.5: In this example, we illustrate the relation of the integral solutions Sol_p with an optimal fractional solution Z_{LP} in terms of moving facilities. We have 2 facilities ($K = 2$) and $N = 3$. The top figure depicts an instance where Z_{LP} has placed $1/N = 1/3$ amount of facility on some nodes in the locations path P. We show in Lemma 10 that Z_{LP} sends these positive fractions of facilities to nodes of the next stage as depicted by the arrows. The bottom figure illustrates how $\mathrm{Sol}_1, \mathrm{Sol}_2$ and Sol_3 move facilities between stages. Observe that each facility is moved to the same node that has moved it's corresponding fraction in Z_{LP}.

The key idea for proving the second claim of Theorem 7 resides again on the structure of the solutions Sol_p. Assume that the optimal fractional solution Z_{LP} places positive $1/N$ amounts of facility m on N nodes of V_{t-1}^+ at stage $t-1$. Then, this set of nodes N^{t-1} are consecutive nodes of V_{t-1}^+ and each Sol_p places facility m on a unique node of N^{t-1}. By solving a flow problem, we show that the optimal fractional solution moves the $1/N$ amounts of facility m to the nodes N^t of V_t^+ in the same way that the solutions Sol_p move the facility m to a node of N^t. Thus, if Z_{LP} moves $1/N$ amount of facility m from node $j_{t-1} \in N^{t-1}$ to node $j_t \in N^t$, there is a unique solution Sol_p that moves facility m from j_{t-1} to j_t. Summing over all facilities and stages, we get that the average moving cost paid by the solutions Sol_p is the same with the moving cost of the optimal fractional solution. The idea behind this proof is depicted in Figure 2.5. The formal proof is presented in Lemma 10.

The second step for proving Theorem 7 is presented in Section 2.2.6, where we show how to generalize the result of Section 2.2.5 for an optimal fractional solution that does not satisfy Assumption 1. Assume that we have such an optimal fractional solution Z_{LP} of the $\mathrm{LP}_{\mathrm{Rllctn}}$ in the locations path P. From this solution, we can construct an optimal fractional solution $\mathrm{Z}_{\mathrm{LP}}^{'}$ of the $\mathrm{LP}_{\mathrm{Rllctn}}$ in a path P' that satisfies Assumption 1. P' is constructed from the locations path P by splitting each node in KN copies of zero distance. Then, in Lemma 11 we show that Z_{LP} and $\mathrm{Z}_{\mathrm{LP}}^{'}$ have the same cost.

We conclude Section 2.2.2 by providing the proof of Theorem 7. By the construction of path P', it is easy to see that the solutions Sol_p correspond to Sol_p^* in the original locations path P with the same connection and moving cost. Since Theorem 7 holds for the solutions Sol_p and $\mathrm{Cost}(\mathrm{Z}_{\mathrm{LP}}) = \mathrm{Cost}(\mathrm{Z}_{\mathrm{LP}}^{'})$, both claims of Theorem 7 hold also for the solutions Sol_p^*. Finally, Sol_1^* is the solution produced by Algorithm 7 therefore Theorem 7 holds in the general case.

Remark 1. *An optimal solution for the K-Facility Reallocation problem on the line can be computed by finding an optimal extreme point for the* $\mathrm{LP}_{\mathrm{Rllctn}}$ *and by proving that this extreme point will be integral. However, proving Theorem 7 serves our end and provides better exposition and brevity.*

2.2.5 Proving Theorem 7 under Assumption 1

Throughout this section, we assume that Assumption 1 is satisfied; f_{kj}^t and c_j^t are either $1/N$ or 0 , for some positive integer N.

The following lemma states that the average connection cost an agent pays in an integral solution Sol_p, at each stage t, is the same with the connection cost that he pays in an optimal fractional solution at each stage t.

Lemma 9. *For an agent i at stage t, under Assumption 1 it holds that*

$$\frac{1}{N}\sum_{p=1}^{N} \text{ConCost}_i^t(\text{Sol}_p) = \sum_{j\in V} \text{d}(\text{Loc}(i,t),j)\zeta_{ij}^t.$$

By Assumption 1, c_j is $1/N$ if $j \in V_t^+$ and 0 otherwise. As a result, in an optimal fractional solution, each agent i finds the N closest to $\text{Loc}(i,t)$ nodes of V_t^+ and receives a $1/N$ amount of service from each one of them. Let us call this set N_i^t. Then, the nodes in N_i^t are consecutive nodes of V_t^+ i.e. $N_i^t = \{Y_l^t, \ldots, Y_{l+N-1}^t\}$ and since $\sum_{j\in V}\zeta_{ij}^t = 1$ we have that

$$\sum_{j\in V} \text{d}(\text{Loc}(i,t),j)\zeta_{ij}^t = \sum_{j=l}^{l+N-1} \text{d}(\text{Loc}(i,t),Y_j^t)/N.$$

Since Sol_p puts facilities in the positions $\{Y_{(m-1)\cdot N+p}^t\}_{m=1}^K$, there exists a unique node $Y_{l(p)}^t \in N_i^t$ on which Sol_p puts a facility. $Y_{l(p)}^t$ is the closest node to $\text{Loc}(i,t)$ from all the nodes in which Sol_p puts a facility. As a result, $\text{ConCost}_i^t(\text{Sol}_p) = \text{d}(\text{Loc}(i), Y_{l(p)}^t)$. Now, summing over p we get,

$$\begin{aligned}
\frac{1}{N}\sum_{p=1}^{N} \text{ConCost}_i^t(\text{Sol}_p) &= \frac{1}{N}\sum_{p=1}^{N} \text{d}(\text{Loc}(i), Y_{l(p)}^t) \\
&= \sum_{j=l}^{l+N-1} \text{d}(\text{Loc}(i), Y_j^t)/N \\
&= \sum_{j\in V} \text{d}(\text{Loc}(i,t),j)\zeta_{ij}^t.
\end{aligned}$$

The proof of Lemma 9 establishes the first claim (1) of Theorem 7. The next step is to prove that the average moving cost of the integral solutions Sol_p is the same with the moving cost of Z_{LP}. This is formally stated in the following lemma.

Lemma 10. *For a facility k at stage t, under Assumption 1 it holds that*

$$\frac{1}{N}\sum_{p=1}^{N} \text{MovCost}(\text{Sol}_p) = \sum_{t=1}^{T}\sum_{k\in F} S_k^t.$$

By Assumption 1, $c_j^t = 1/N$ if $j \in V_t^+ = \{Y_1^t, \ldots, Y_{KN}^t\}$ and 0 otherwise. Notice that the connection cost of the optimal fractional solution only depends on the variables c_j^t. As a result, $f_{kj}^t, S_k^t, S_{kjl}^t$ must be the optimal solution of the following linear program.

$$
\begin{array}{lll}
\text{minimize} & \sum_{t=1}^{T}\sum_{k=1}^{K} S_k^t & \\
\text{s.t.} & \sum_{k\in[K]} f_{kj}^t = \frac{1}{N} & \forall j \in V_t^+, t\in[T] \\
& \sum_{j\in V_t^+} f_{kj}^t = 1 & \forall k\in[K], t\in[T] \\
& S_k^t = \sum_{j,l\in V} d(j,l) S_{kjl}^t & \forall k\in[K], t\in[T] \\
& \sum_{j\in V_{t-1}^+} S_{kjl}^t = f_{kl}^t & \forall k\in[K], l\in V_t^+, t\in[T] \\
& \sum_{l\in V_t^+} S_{kjl}^t = f_{kj}^{t-1} & \forall k\in[K], j\in V_{t-1}^+, t\in[T]
\end{array}
\tag{2.2}
$$

We can relax the linear program above (2.2) to get the more convenient linear program that follows.

$$
\begin{array}{lll}
\text{minimize} & \sum_{t=1}^{T}\sum_{j\in V_{t-1}^+, l\in V_t^+} d(j,l) F_{jl}^t & \\
\text{s.t.} & \sum_{l\in V_t^+} F_{jl}^t = \frac{1}{N} & \forall j\in V_{t-1}^+, t\in[T] \\
& \sum_{j\in V_{t-1}^+} F_{jl}^t = \frac{1}{N} & \forall l\in V_t^+, t\in[T]
\end{array}
\tag{2.3}
$$

It is easy to prove that the LP (2.3) is a relaxation of LP (2.2) by setting $F_{jl}^t = \sum_{k\in F} S_{kjl}^t$. Moreover, the above LP describes a flow problem between the nodes V_t^+, where F_{jl}^t is the amount of flow going from node $j \in V_{t-1}^+$ to node $l \in V_t^+$ (see Figure 2.6).

Next, we proceed to the final step of our proof. First, observe that $F_{Y_j^{t-1}Y_j^t}^t$ is feasible solution for the above LP since $|V_{t-1}^+| = |V_t^+| = K \cdot N$. If we prove that this assignment minimizes the objective, then we are done. Assume that in an optimal solution, $F_{Y_1^{t-1}Y_1^t}^t < 1/N$. Since $\sum_{l\in V_t^+} F_{Y_1^{t-1}l}^t = \frac{1}{N}$, there exists Y_j^t such that $F_{Y_1^{t-1}Y_j^t}^t > 0$. Similarly, by using the second constraint we obtain that $F_{Y_{j'}^{t-1}Y_1^t}^t > 0$. Let $\epsilon = \min(F_{Y_1^{t-1}Y_j^t}^t, F_{Y_{j'}^{t-1}Y_1^t}^t)$. Observe that if we increase $F_{Y_1^{t-1}Y_1^t}^t, F_{Y_{j'}^{t-1}Y_j^t}^t$ by ϵ and decrease $F_{Y_1^{t-1}Y_j^t}^t, F_{Y_{j'}^{t-1}Y_1^t}^t$ by ϵ, we obtain another feasible solution. The cost difference of the two solutions is $D = \epsilon(d(Y_1^{t-1}, Y_j^t) + d(Y_{j'}^{t-1}, Y_1^t) - d(Y_1^{t-1}, Y_1^t) - d(Y_{j'}^{t-1}, Y_j^t))$. If we prove that D is non-negative, we are done. We show the latter using the fact that $Y_1^{t-1} \le Y_{j'}^{t-1}$ and $Y_1^t \le Y_j^t$. More precisely,

- If $Y_1^{t-1} \le Y_1^t$ then $D \ge 0$ since $Y_1^t \le Y_j^t$.
- If $Y_1^{t-1} \ge Y_1^t$ then $D \ge 0$ since $Y_1^{t-1} \le Y_{j'}^{t-1}$.

Until now, we have shown that in the optimal solution, the node Y_1^{t-1} sends all of her flow to the node Y_1^t. Meaning that Y_1^t does not receive flow by any other node apart from Y_1^{t-1}. By repeating the same argument, it follows that in the optimal solution each node Y_j^{t-1} sends all of her flow to Y_j^t. Thus, we have that

$$\frac{1}{N}\sum_{t=1}^{T}\sum_{j=1}^{K\cdot N} \mathrm{d}(Y_j^{t-1}, Y_j^t) = \sum_{t=1}^{T}\sum_{k\in F} S_k^t. \tag{2.4}$$

We can see an illustration of the previous arguments below:

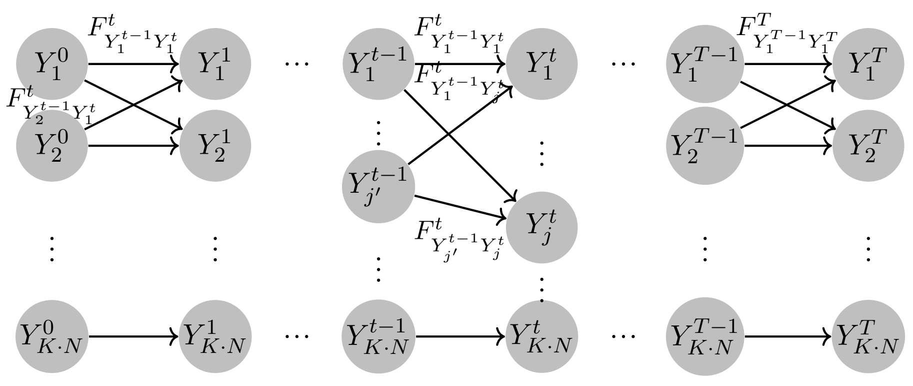


Figure 2.6: The flow described by LP (2.3).

Finally, by the definition of the solutions Sol_p we have that:

$$\begin{aligned}
\frac{1}{N}\sum_{p=1}^{N}\mathrm{MovCost}(\mathrm{Sol}_p) &= \frac{1}{N}\sum_{p=1}^{N}\sum_{t=1}^{T}\sum_{m=1}^{K}\mathrm{d}(Y^{t-1}_{(m-1)N+p}, Y^{t}_{(m-1)N+p})\\
&= \frac{1}{N}\sum_{t=1}^{T}\sum_{m=1}^{K}\sum_{p=1}^{N}\mathrm{d}(Y^{t-1}_{(m-1)N+p}, Y^{t}_{(m-1)N+p})\\
&= \frac{1}{N}\sum_{t=1}^{T}\sum_{j=1}^{K\cdot N}\mathrm{d}(Y^{t-1}_{j}, Y^{t}_{j})\\
&= \sum_{t=1}^{T}\sum_{k\in F} S^t_k,
\end{aligned}$$

where the the last equality follows from equation 2.4.

The previous lemma concludes the proof of Theorem 7 under Assumption 1. In the next section, we show how to prove Theorem 7 in the general case.

2.2.6 Proving Theorem 7 in the General Case

In this section, we prove Theorem 7 in the general case. This will be achieved in the following steps:

1. From an optimal fractional solution $\mathrm{Z_{LP}}$ for the original path P, we will construct an optimal fractional solution $\mathrm{Z}^{'}_{\mathrm{LP}}$ for a path P' that satisfies Assumption 1.
2. We will show that $\mathrm{Z_{LP}}$ and $\mathrm{Z}^{'}_{\mathrm{LP}}$ have the same cost.
3. We will define integral solutions Sol^*_p (see Definition 6) in P that correspond to the integral solutions Sol_p in P' that satisfy Assumption 1.
4. The solution provided by Algorithm 7 is one of the solutions Sol^*_p.

We begin by constructing an optimal fractional solution that satisfies Assumption 1 from a generic optimal fractional solution. As already discussed, Assumption 1 is not satisfied in general by a fractional solution of the relaxation of the $\mathrm{LP_{Rllctn}}$ (2.2). Each $S^t_{kj\ell}$ will be either 0 or $A^t_{kj\ell}/N^t_{kj\ell}$, for some positive integers $A^t_{kj\ell}, N^t_{kj\ell}$. However, each positive f^t_{kj} has the form B^t_{kj}/N, where $N = \Pi_{S^t_{kj\ell}>0}N^t_{kj\ell}$. This is due to the constraint $f^t_{kj} = \sum_{j\in V} S^t_{kj\ell}$.

Now, consider the path $P' = (V', E')$ constructed from path $P = (V, E)$ as follows: Each node $j \in V$ is split into KN copies $\{j_1, \ldots, j_{KN}\}$ with zero distance between them. Consider also the

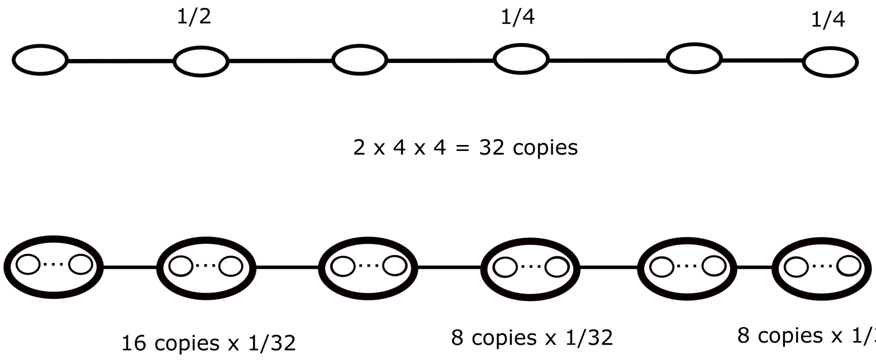


Figure 2.7: In this figure, we see on the top a fractional solution for the original locations path P and on the bottom a fractional solution for the path P' under Assumption 1. The instance has one facility ($K = 1$). In P, we have an amount of $c_j^t = 1/2$ on the second node from the left and an amount of $c_j^t = 1/4$ on the fourth and sixth node on the left. Therefore $N = 2 \cdot 2 \cdot 4 = 32$ and each node of P splits in 32 copies with zero distance in P'. Thus, in P', we have 16 nodes corresponding to the second node in P with $c_j^t = 1/32$, 8 nodes corresponding to the fourth node in P with $c_j^t = 1/32$ and 8 nodes corresponding to the sixth node in P with $c_j^t = 1/32$. Thus, these two solutions have the same cost.

LP$_{\text{Rllctn}}$ (2.2), when the underlying path is $P' = (V', E')$ and at each stage t, each agent i is located at a node of V' that is a copy of i's original location, $\text{Loc}'(i,t) = \ell \in V'$, where $\ell \in \text{Copies}(\text{Loc}(i,t))$. Even though paths P and P' form two different linear programs, these linear programs are closely related since a solution for one can be converted to a solution for the other with the exact same cost. This is due to the fact that for all $j, h \in V$, $\text{d}(j,h) = \text{d}(j',h')$, where $j' \in \text{Copies}(j)$ and $h' \in \text{Copies}(h)$.

Given the fractional positions $\{f_{kj}^t\}_{t\geq 1}$ of an optimal solution of the LP formulated for $P = (V,E)$, we construct the fractional positions of the facilities in $P' = (V', E')$ as follows: If $f_{kj}^t = B_{kj}^t/N$, then facility k puts a $1/N$ amount of facility in B_{kj}^t nodes of the set $\text{Copies}(j) = \{j_1, \dots, j_{KN}\}$ that have a 0 amount of facility. The latter is possible since there are exactly KN copies of each $j \in V$ and $c_j^t \leq K$ (that is the reason we required KN copies of each node). The values for the rest of the variables are defined in the proof of the following lemma.

Lemma 11. *Let $\{f_{kj}^t, c_j^t, S_{kjl}^t, \zeta_{ij}^t\ S_k^t\}_{t\geq 1}$ be an optimal fractional solution Z_{LP} for the LP (2.2) in the locations path P. Then, there exists a solution $\text{Z}_{\text{LP}}' = \{f_{kj}'^t, c_j'^t, S_{kjl}'^t, \zeta_{ij}'^t, S_k'^t\}_{t\geq 1}$ of the LP (2.2) in the path P' such that:*

- $\text{Cost}(\text{Z}_{\text{LP}}')= \text{Cost}(\text{Z}_{\text{LP}})$ *(1)*
- $f_{k\ell}'^t = 1/N$ *or* 0, *for each* $\ell \in V'$ *(2)*
- $c_\ell'^t = 1/N$ *or* 0, *for each* $\ell \in V'$ *(3)*
- $c_j^t = \sum_{\ell \in \textit{Copies}(j)} c_\ell'^t$, *for each* $j \in V'$ *(4)*

We start by setting values to the variables $f_{kj}'^t$. Initially, all $f_{kj}'^t = 0$. We know that if $f_{kj}^t > 0$, then it equals B_{kj}^t/N, for some positive integer B_{kj}^t. For each such f_{kj}^t, we find $u_1, \dots, u_{B_{kj}^t} \in \text{Copies}(j)$ with $f_{ku_h}'^t = 0$. Then, we set $f_{ku_h}'^t = 1/N$ for $h = \{1, \dots, B_{kj}^t\}$. Since there are KN copies of each node $j \in V$ and $\sum_{j\in V} f_{kj}^t \leq K$, we can always find sufficient copies of j with $f_{ku}'^t = 0$. When this step is terminated, we are sure that conditions $2, 3, 4$ are satisfied.

We continue with the variables $S_{kj\ell}'^t$. Initially, all $S_{kj\ell}'^t = 0$. Then, each positive $S_{kj\ell}^t$ has the form $B_{kj\ell}^t/N$. Let $B = B_{kj\ell}^t$ to simplify notation. We now find B copies of $u_1, \dots, u_B$ of j and $v_1, \dots, v_B$ of ℓ so that

- $f'^t_{ku_1} = \ldots = f'^t_{ku_B} = f'^t_{kv_1} = \ldots = f'^t_{kv_B} = 1/N$.
- $S'^t_{ku_1h} = \ldots = S'^t_{ku_Bh} = S'^t_{khv_1} = \ldots = S'^t_{khv_B} = 0$ for all $h \in V'$.

We then set $S'^t_{ku_1v_1} = \ldots = S'^t_{ku_Bv_B} = 1/N$. Since $\sum_{\ell\in V} S^t_{kj\ell} = f^t_{kj}$ and $\sum_{j\in V} S^t_{kj\ell} = f^t_{k\ell}$, we can always find $B^t_{kj\ell}$ pairs of copies of j and ℓ that satisfy the above requirements. We can now prove that the movement cost of each facility k is the same in both solutions.

$$\begin{aligned}\sum_{j\in V}\sum_{\ell\in V} \mathrm{d}(j,\ell)S^t_{kj\ell} &= \sum_{j\in V}\sum_{\ell\in V} \mathrm{d}(j,\ell)B^t_{kj\ell}/N\\ &= \sum_{j\in V}\sum_{\ell\in V}\sum_{h\in \mathrm{Copies}(j)}\sum_{h'\in\mathrm{Copies}(\ell)} S'^t_{khh'}\mathrm{d}(h,h')\\ &= \sum_{j'\in V'}\sum_{\ell'\in V'} S'^t_{kj'\ell'}\mathrm{d}(j',\ell').\end{aligned}$$

The second equality follows from the fact that h and h' are copies of j and ℓ respectively and thus $\mathrm{d}(h,h') = \mathrm{d}(j,\ell)$.

Finally, set values to the variables ζ'^t_{ij} for each $j \in V'$. Again, each positive ζ^t_{ij} equals B^t_{ij}/N, for some positive integer. We take B^t_{ij} copies of $j, u_1, \ldots, u_{B^t_{ij}}$ and set $\zeta'^t_{iu_1} = \ldots = \zeta'^t_{iu_{B^t_{ij}}} = 1/N$. The connection cost of each agent i remains the same since

$$\begin{aligned}\sum_{j\in V}\mathrm{d}(\mathrm{Loc}(i,t),j)\zeta^t_{ij} &= \sum_{j\in V}\mathrm{d}(\mathrm{Loc}(i,t),j)B^t_{ij}/N\\ &= \sum_{j\in V}\mathrm{d}(\mathrm{Loc}(i,t),j)\sum_{j'\in\mathrm{Copies}(j)}\zeta'^t_{ij'}\\ &= \sum_{j\in V}\sum_{j'\in\mathrm{Copies}(j)}\mathrm{d}(\mathrm{Loc}'(i,t),j')\zeta'^t_{ij'}\\ &= \sum_{h\in V'}\mathrm{d}(\mathrm{Loc}'(i,t),h)\zeta'^t_{ih},\end{aligned}$$

where the third equality holds since $\mathrm{Loc}'(i,t) \in \mathrm{Copies}(\mathrm{Loc}(i,t))$.

Before, we move to the proof of our main result, we give the formal definition of the integral solutions Sol^*_p for the original path P (these are referred as Sol_p in Theorem 7).

Definition 6. Sol^*_p *denotes the integral solution in the original positions path P that corresponds to the solution* Sol_p *of path P'. That is, if* Sol^*_p *places the mth facility to a node $j \in V$ in P, then* Sol_p *places the mth facility to a node j_c in P' that is one of the copies of j, namely $j_c \in$* $\mathrm{Copies}(j)$.

To conclude this section, we prove Theorem 7.

Proof of Theorem 7. By Lemma 11, an optimal fractional solution $\mathrm{Z}'_{\mathrm{LP}}$ for P' satisfies Assumption 1. Applying Lemma 9 for $\mathrm{Z}'_{\mathrm{LP}}$, we get N integral solutions Sol_p for P', which place the mth facility to the $(m-1)N+p$, $1 \le p \le N$, node of V^+_t ($Y^t_{(m-1)N+p} \in V'$) such that

$$\frac{1}{N}\sum_{p=1}^{N}\mathrm{ConCost}^t_i(\mathrm{Sol}_p) = \sum_{j\in V'}\mathrm{d}(\mathrm{Loc}(i,t),j)\zeta'^t_{ij}$$

where the right hand side of the equality is the connection cost for the fractional solution $\mathrm{Z}'_{\mathrm{LP}}$ in P'. By Lemma 11 the connection cost of the solution $\mathrm{Z}'_{\mathrm{LP}}$ for P' equals the connection cost of an optimal fractional solution $\mathrm{Z}'_{\mathrm{LP}}$ for the original path P. By Definition 6, the integral solution Sol^*_p places facility m, $1 \leq m \leq K$, on a node in P which has zero distance from the node that the integral solution Sol_p places the facility m. Thus, we have N integral solutions Sol^*_p for P, such that

$$\frac{1}{N}\sum_{p=1}^{N} \mathrm{ConCost}_i^t(\mathrm{Sol}^*_p) = \sum_{j \in V} \mathrm{d}(\mathrm{Loc}(i,t), j)\zeta_{ij}^t.$$

This proves the first claim (1) of Theorem 7. Using the same arguments and Lemma 10, we can prove that the average moving cost of the solutions Sol^*_p equals the moving cost of the optimal fractional solution Z_{LP}. Thus, both claims Theorem 7 hold. Finally, it is easy to see that the solution provided by Algorithm 1 corresponds to Sol^*_1, thus is one of the integral solutions for P and this concludes the proof of Theorem 7. □

Chapter 3

Approximation Algorithms for Multistage Min Sum Set Cover

3.1 Preliminaries and Basic Definitions

The set of elements e is denoted by U with $|U| = n$. A permutation of the elements is denoted by π where π_i denotes the element lying at position i (for $1 \leq i \leq n$) and $\text{Pos}(e, \pi)$ denotes the position of the element $e \in U$ in permutation π.

Definition 7 (Kendall-Tau Distance). *Given the permutations π^A, π^B, a pair of elements (e, e') is inverted if and only if $\text{Pos}(e, \pi^A) > \text{Pos}(e', \pi^A)$ and $\text{Pos}(e, \pi^B) < \text{Pos}(e', \pi^B)$ or vice versa. The Kendall-Tau distance between the permutations π^A, π^B, denoted by $\text{d}_{\text{KT}}(\pi^A, \pi^B)$, is the number of inverted pairs.*

Definition 8 (Spearman' Footrule Distance). *The FootRule distance between the permutations π^A, π^B is defined as $\text{d}_{\text{FR}}(\pi^A, \pi^B) = \sum_{e \in U} |\text{Pos}(e, \pi^A) - \text{Pos}(e, \pi^B)|$.*

The Kendall-Tau distance and FootRule distance are approximately equivalent, $\text{d}_{\text{KT}}(\pi^A, \pi^B) \leq \text{d}_{\text{FR}}(\pi^A, \pi^B) \leq 2 \cdot \text{d}_{\text{KT}}(\pi^A, \pi^B)$. Moreover both of them satisfy the triangle inequality.

Definition 9. *An $n \times n$ matrix with positive entries (rows stand for the elements and columns for the positions) is called stochastic if $\sum_{i=1}^{n} A_{ei} = 1$ for all $e \in U$ and doubly stochastic if (additionally) $\sum_{e \in U} A_{ei} = 1$ for all $1 \leq i \leq n$.*

A permutation of the elements π can be equivalent represented by a 0-1 doubly stochastic matrix A, where $A_{ei} = 1$ if element e lies at position i and 0 otherwise. When clear from context, we use the notion of permutation and (0-1) doubly stochastic matrix interchangeably.

The notion of FootRule distance can be naturally extended to stochastic matrices. Given two doubly stochastic matrices A, B consider the min-cost transportation problem, transforming row A_e to the row B_e where the cost of transporting a unit of mass between column i and column j equals $|i - j|$. Formally for each row e, define a complete bipartite graph where on the left part lie the entries (e, i) for $1 \leq i \leq n$ and on the right part the entries (e, j) for $1 \leq j \leq n$. The mass transported from entry (e, i) to entry (e, j) (denoted as f_{ij}^e) costs $f_{ij}^e \cdot |i - j|$ and the total mass *leaving* (e, i) equals A_{ei} and the total mass *arriving* at (e, j) equals B_{ej}.

Definition 10. *The FootRule distance between two stochastic matrices A, B, denoted by $\text{d}_{\text{FR}}(A, B)$, is the optimal value of the following linear program,*

$$\begin{aligned}
\textit{min} \quad & \sum_{e \in U} \sum_{i=1}^{n} \sum_{j=1}^{n} |i - j| \cdot f_{ij}^e \\
\textit{s.t} \quad & \sum_{i=1}^{n} f_{ij}^e = B_{ej} \quad \textit{for all } e \in U \textit{ and } j = 1, \ldots, n \\
& \sum_{j=1}^{n} f_{ij}^e = A_{ei} \quad \textit{for all } e \in U \textit{ and } i = 1, \ldots, n \\
& f_{ij}^e \geq 0 \qquad \textit{for all } e \in U \textit{ and } i, j = 1, \ldots, n
\end{aligned}$$

Let the stochastic matrices $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 1/3 & 1/3 & 1/3 \\ 1/2 & 1/2 & 0 \\ 1/4 & 0 & 3/4 \end{pmatrix}$. The FootRule distance $\mathrm{d}_{\mathrm{FR}}(A, B) = \underbrace{(0 \cdot 1/3 + 1 \cdot 1/3 + 2 \cdot 1/3)}_{\text{first row}} + \underbrace{(1 \cdot 1/2 + 0 \cdot 1/2 + 1 \cdot 0)}_{\text{second row}} + \underbrace{(2 \cdot 1/4 + 1 \cdot 0 + 0 \cdot 3/4)}_{\text{third row}} = 2$.

Up next we present the formal definition of Multistage Min-Sum Set Cover.

Definition 11 (Multistage Min-Sum Set Cover). *Given a universe of elements U, a sequence of requests $R_1, \ldots, R_T \subseteq U$ and an initial permutation of the elements π^0. The goal is to select a sequence of permutation $\pi^1, \ldots, \pi^T$ that minimizes*

$$\sum_{t=1}^{T} \pi^t(R_t) + \sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})$$

where $\pi^t(R_t)$ is the position of the first element of R_t that we encounter in π^t, $\pi^t(R_t) = \min\{1 \leq i \leq n : \pi^t_i \in R_t\}$.

We refer to $\sum_{t=1}^{T} \pi^t(R_t)$ as **covering cost** and to $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})$ as **moving cost**. We denote with π^t_{Opt} the permutation of the optimal solution of Mult-MSSC at round t, with o_t the element that the optimal solution uses to cover the request R_t (the element of R_t appearing first in π^t_{Opt}), and with $\mathrm{OPT}_{\text{Mult-MSSC}}$ the cost of the optimal solution. Finally we call an instance of Mult-MSSC *r-bounded* in case the cardinality of the requests is bounded by r, $|R_t| \leq r$.

3.2 Approximation Algorithms for Dynamic Min-Sum Set Cover

There exists an approximation-preserving reduction from Set – Cover to Mult-MSSC that provides us with the following inapproximability results.

Theorem 8.
- *There is no $c \cdot \log n$-approximation algorithm for* Mult-MSSC *(for a sufficiently small constant c) unless* P = NP.
- *For r-bounded sequences, there is no $o(r)$-approximation algorithm for* Mult-MSSC*, unless there is a $o(r)$-approximation algorithm for* Set – Cover *with each element being covered by at most r sets.*

The proof of Theorem 8 is fairly simple, given an instance of Set – Cover we construct an instance of Mult-MSSC in which the initial permutation π^0 contains in the first positions some dummy elements (they do not appear in any of the requests) and in the last positions the sets of the Set – Cover (we consider an element of Mult-MSSC for each set of Set – Cover). Finally each request for Mult-MSSC is associated with an element of the Set – Cover and contains the *elements in* Mult-MSSC/ *sets in* Set – Cover containing it.

Proof. Let the equivalent definition of Set – Cover in which we are given a universe of element $E = \{1, \ldots, n\}$ and sets $S_1, S_2, \ldots, S_m \subseteq E$ and we are asked to select the minimum number of elements covering all the sets (an element e covers set S_i if $e \in S_i$).

Consider the instance of Mult-MSSC with the elements $U = \{1, \ldots, n\} \cup \{d_1, \ldots, d_{n^2 m}\}$. The elements $\{d_1, \ldots, d_{n^2 m}\}$ are dummy in the sense that they appear in none of the requests R_t. Let the initial permutation π_0 contain in the first $n^2 m$ positions the dummy elements and in the last n positions the elements $\{1, \ldots, n\}$, $\pi_0 = [d_1, \ldots, d_{n^2 m}, 1, \ldots, n]$ and the request sequence of Mult-MSSC be $S_1, S_2, \ldots, S_m$.

Let a c-approximation algorithm for Mult-MSSC producing the permutation $\pi_1, \dots, \pi_m$ the cost of which is denoted by Alg. Let also CoverAlg denote the set composed by the element that the c-approximation algorithm uses to cover the requests, CoverAlg $= \{$the element of S_t appearing first in $\pi_t\}$. Then,

$$\text{Alg} \geq n^2 m \cdot |\text{CoverAlg}|$$

Now consider the following solution for Mult-MSSC constructed by the optimal solution for Set $-$ Cover. This solution initially moves the elements of the optimal covering set $\text{OPT}_{\text{SetCover}}$ to the first positions and then never changes the permutation. Clearly the cost of this solution is upper bounded by

$$\text{Set} - \text{Cover}_{\text{Mult-MSSC}} \leq \underbrace{|\text{OPT}_{\text{SetCover}}| \cdot (n^2 m + n)}_{\text{moving cost}} + \underbrace{m \cdot |\text{OPT}_{\text{SetCover}}|}_{\text{covering cost}}$$

In case Alg $\leq c \cdot$ Set $-$ $\text{Cover}_{\text{Mult-MSSC}}$, we directly get that $|\text{CoverAlg}| \leq 3c \cdot |\text{OPT}_{\text{SetCover}}|$.

There is no polynomial-time approximation algorithm for Set $-$ Cover with approximation ratio better than $\log m$. The latter holds even for instance of Set $-$ Cover for which $m = \text{poly}(n)$ [3] where $\text{poly}(\cdot)$ is a polynomial with degree bounded by a universal constant. Since the number of elements $|U|$, in the constructed instance of Mult-MSSC is $n^2 m$, any $c \cdot \log |U|$-approximation for Mult-MSSC (for c sufficiently small) implies an approximation algorithm for Set $-$ Cover with approximation ratio less than $\log n$. In case there exists an $c = o(r)$-approximation algorithm for Mult-MSSC for requests sequences $R_1, \dots, R_T$ where $|R_t| \leq r$, we obtain an $o(r)$-approximation for algorithm for Set $-$ Cover for sets with cardinality bounded by r. In the standard form of Set $-$ Cover this is translated into the fact that each element belongs in at most r sets. ☐

Both the $O(\log^2 n)$-approximation algorithm (for requests of general cardinality) and the $O(r^2)$-approximation algorithm for r-bounded requests, that we subsequently present, are based on rounding a linear program called *Fractional Move To Front*. The latter is the linear program relaxation of *Move To Front*, a problem closely related to Multistage Min-Sum Set Cover. MTF asks for a sequence of permutations $\pi^1, \dots, \pi^T$ such as at each round t, an element of R_t lies on the first position of π^t and $\sum_{t=1}^{T} \text{d}_{\text{FR}}(\pi^t, \pi^{t-1})$ is minimized.

Definition 12. *Given a sequence of requests $R_1, \dots, R_T \subseteq U$ and an initial permutation of the elements π^0, consider the following linear program, called* Fractional $-$ MTF*,*

$$\begin{array}{lll}
\textit{min} & \sum_{t=1}^{T} \text{d}_{\text{FR}}(A^t, A^{t-1}) & \\
\textit{s.t} & \sum_{i=1}^{n} A^t_{ei} = 1 & \textit{for all } e \in U \textit{ and } t = 1, \dots, T \\
& \sum_{e \in U} A^t_{ei} = 1 & \textit{for all } i = 1, \dots, n \textit{ and } t = 1, \dots, T \\
& \sum_{e \in R_t} A^t_{e1} = 1 & \textit{for all } t = 1, \dots, T \\
& A^0 = \pi^0 & \\
& A^t_{ei} \geq 0 & \textit{for all } e \in U,\ i = 1, \dots, n \textit{ and } t = 1, \dots, T
\end{array}$$

where $\text{d}_{\text{FR}}(\cdot, \cdot)$ *is the FootRule distance of Definition 10.*

There is an elegant argument (appeared in previous works, e.g., [28]) showing that the optimal solution of MTF is at most $4 \cdot \text{OPT}_{\text{Mult-MSSC}}$. In Lemma 12 we provide the argument and establish that Fractional $-$ MoveToFront is a 4-approximate relaxation of Mult-MSSC.

Lemma 12. $\sum_{t=1}^{T} \text{d}_{\text{FR}}(A^t, A^{t-1}) \leq 4 \cdot \text{OPT}_{\text{Mult-MSSC}}$ *where $A^1, \dots, A^t$ is the optimal solution of* Fractional $-$ MTF.

Proof of Lemma 12. Let o_t the element of R_t appearing first in the permutation π^t_{Opt}. Consider the sequence of permutation $\pi^0, \pi^1, \dots, \pi^T$ constructed by moving at each round t, the element o_t to the first position of the permutation. Notice that $\pi^0, \pi^1, \dots, \pi^T$ is a feasible solution for both MoveToFront and Fractional − MTF. The first key step towards the proof of Lemma 12 is that

$$\mathrm{d_{KT}}(\pi^t, \pi^{t-1}) + \mathrm{d_{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d_{KT}}(\pi^{t-1}, \pi^t_{\text{Opt}}) \le 2 \cdot \pi^t_{\text{Opt}}(R_t)$$

To understand the above inequality, let k_t be the position of o_t in permutation π^{t-1}. Out of the $k_t - 1$ elements on the right of o_t in permutation π^{t-1}, let $Left_t$ ($Right_t$) denote the elements that are on the left (right) of o_t in permutation π^{t-1}_{Opt}. It is not hard to see that $\pi^t_{\text{Opt}}(R_t) \ge |Left_t|$, $\mathrm{d_{KT}}(\pi^t, \pi^{t-1}) = |Left_t| + |Right_t|$ and $\mathrm{d_{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d_{KT}}(\pi^{t-1}, \pi^t_{\text{Opt}}) = |Left_t| - |Right_t|$. Using the fact that $\mathrm{d_{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d_{KT}}(\pi^{t-1}, \pi^t_{\text{Opt}}) \le \mathrm{d_{KT}}(\pi^t_{\text{Opt}}, \pi^{t-1}_{\text{Opt}})$ and the previous inequality we get,

$$\mathrm{d_{KT}}(\pi^t, \pi^{t-1}) + \mathrm{d_{KT}}(\pi^t, \pi^t_{\text{Opt}}) - \mathrm{d_{KT}}(\pi^{t-1}, \pi^{t-1}_{\text{Opt}}) \le 2 \cdot \pi^t_{\text{Opt}}(R_t) + \mathrm{d_{KT}}(\pi^t_{\text{Opt}}, \pi^{t-1}_{\text{Opt}})$$

and by a telescopic sum we get $\sum_{t=1}^T \mathrm{d_{KT}}(\pi^t, \pi^{t-1}) \le 2 \cdot \text{OPT}_{\text{Mult-MSSC}}$. The proof follows by the fact that $\mathrm{d_{FR}}(\pi^t, \pi^{t-1}) \le 2 \cdot \mathrm{d_{KT}}(\pi^t, \pi^{t-1})$. □

As already mentioned, our main technical contribution is the design of *rounding schemes* converting the optimal solution, $A^1, \dots, A^T$, of Fractional − MTF into a sequence of permutations $\pi^1, \dots, \pi^T$. This is done so as to bound the moving cost of our algorithms by the moving cost $\sum_{t=1}^T \mathrm{d_{FR}}(A^t, A^{t-1})$. We then separately bound the covering cost, $\sum_{t=1}^T \pi^t(R_t)$ by showing that always an element of R_t lies on the first positions of π^t.

The main technical challenge in the design of our rounding schemes is ensure to that the moving cost of our solutions $\sum_{t=1}^T \mathrm{d_{KT}}(\pi^t, \pi^{t-1})$ is approximately bounded by the moving cost $\sum_{t=1}^T \mathrm{d_{FR}}(A^t, A^{t-1})$. Despite the fact that the connection between doubly stochastic matrices and permutations is quite well-studied and there are various rounding schemes converting doubly stochastic matrices to probability distributions on permutations (such as the Birkhoff–von Neumann decomposition or the schemes of [10, 39, 8, 28]), using such schemes in a *black-box* manner does not provide any kind of positive results for Mult-MSSC. For example consider the case where $A^1 = \cdots = A^T$ and thus $\sum_{t=1}^T \mathrm{d_{FR}}(A^t, A^{t-1}) = \mathrm{d_{FR}}(A^1, A^0)$. In case a randomized rounding scheme is applied *independently to each* A^t, there always exists a positive probability that $\pi^t \neq \pi^{t-1}$ and thus the moving cost will far exceed $\mathrm{d_{FR}}(A^1, A^0)$ as T grows. The latter reveals the need for *coupled rounding schemes* that convert the overall sequence of matrices $A^1, \dots, A^T$ to a sequence of permutations $\pi^1, \dots, \pi^T$. Such a rounding scheme is presented in Algorithm 8 and constitutes the back-bone of our approximation algorithm for requests of general cardinality.

Algorithm 8 A Randomized Algorithm for Mult-MSSC

```
Input: A sequence of requests $R_1, \dots, R_T$ and an initial permutation of the elmenents $\pi^0$.
Output: A sequence of permutations $\pi^1, \dots, \pi^T$.
 1: Find the optimal solution $A^0 = \pi^0, A^1, \dots, A^T$ for Fractional − MTF.
 2: for each element $e \in U$ do
 3:    Select $\alpha_e$ uniformly at random in $[0, 1]$.
 4: end for
 5: for $t = 1 \dots T$ do
 6:    for all elements $e \in U$ do
 7:       $I^t_e := \text{argmin}_{1 \le i \le n}\{\log n \cdot \sum_{s=1}^i A^t_{es} \ge \alpha_e\}$.
 8:    end for
 9:    $\pi^t :=$ sort elements according to $I^t_e$ with ties being broken lexicographically.
10: end for
```

The rounding scheme described in Algorithm 8, imposes correlation between the different time-steps by simply requiring that each element e selects α_e once and for all and by breaking ties lexicographically (any consistent tie-breaking rule would also work). In Lemma 13 of Section 3.3, we show that no matter the sequence of doubly stochastic matrices, the rounding scheme of Algorithm 8 produces a sequence of permutations with overall moving cost at most $4\log^2 n$ the moving cost of the matrix-sequence[1] and thus establishes that the overall moving cost of Algorithm 8 is bounded by $4\log^2 n \cdot \text{OPT}_{\text{Mult-MSSC}}$. The $\log n$ multiplication in Step 7 serves as a *probability amplifier* ensuring that at least one element of R_t lies in the relatively first positions of π^t and permits us to approximately bound the covering cost $\sum_{t=1}^{T} \mathbb{E}\left[\pi^t(R_t)\right]$ by the covering cost of the optimal solution for Mult-MSSC, $\sum_{t=1}^{T} \pi^t_{\text{Opt}}(R_t)$.

Theorem 9. *Algorithm 8 is a $O(\log^2 n)$-approximation algorithm for* Mult-MSSC.

Despite the fact that in Step 7 of Algorithm 8, we multiply the entries of A^t with $\log n$ the overall guarantee is $O(\log^2 n)$. At a first glance the latter seems quite strange but admits a rather natural explanation. For most of the positions i, the probability that an element e admits index $I^t_e = i$ is roughly $\log n \cdot A^t_{ei}$, but due to the fact each index $j \leq i$ is on expectation selected by $\log n$ other elements, the expected position of e in the produced permutation is roughly $\log^2 n$ times the expected value of $\text{argmin}_{1\leq i\leq n}\{\sum_{s=1}^{i} A^t_{es} \geq \alpha_e\}$. This phenomenon relates with the elegant fitting argument given in [24] to prove that the greedy algorithm is 4-approximation for the original *Min-Sum Set Cover* (which is tight unless P $=$ NP). The latter makes us conjecture that the tight inapproximability bound for Mult-MSSC is $\Omega(\log^2 n)$ for requests of general cardinality.

Motivated by the r-approximation LP-based algorithm for instances of Set $-$ Cover in which elements belong in at most r sets, we examine whether the $O(\log^2 n)$ for Mult-MSSC can be ameliorated in case of r-bounded request sequences. Interestingly, the simple *greedy rounding* scheme (described[2] in Algorithm 9) provides such a $O(r^2)$-approximation algorithm.

Algorithm 9 A Greedy-Rounding Algorithm for Mult-MSSC for r-Bounded Sequences.

```
Input: A request sequence $R_1, \dots, R_T$ with $|R_t| \leq r$ and an initial permutation $\pi^0$.
Output: A sequence of permutations $\pi^1, \dots, \pi^T$.
1: Find the optimal solution $A^0 = \pi^0, A^1, \dots, A^T$ for Fractional − MTF.
2: for $t = 1 \dots T$ do
3:     $\pi^t :=$ in $\pi^{t-1}$, move to the first position an element $e \in R_t$ such that $A^t_{e1} \geq 1/r$
4: end for
```

The $O(r^2)$-approximation guarantee of Algorithm 11 is formally stated and proven in Theorem 10. The main technical challenge is that we cannot directly compare the moving cost of Algorithm 9 with $\sum_{t=1}^{T} \text{d}_{\text{FR}}(A^t, A^{t-1})$ and thus we deploy a two-step detour.

In the first step (Lemma 15), we prove the existence of a sequence of doubly stochastic matrices $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ for which each $\hat{A}^t$ satisfies that **(i)** its entries of are **multiples of** $1/r$, **(ii)** $\hat{A}^t_{e_t 1} \geq 1/r$ where e_t is the element that Algorithm 9 moves to the first position at round t, and **(iii)** the sequence $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ admits moving cost at most $\sum_{t=1}^{T} \text{d}_{\text{FR}}(A^t, A^{t-1})$. In order to establish the existence of such a sequence, we construct an appropriate linear program (see Definition 14) based on the elements that Algorithm 9 moves to the first position at each round and prove that it admits an optimal solution with values being multiples of $1/r$. To do the latter, we relate the linear program of Definition 14 with a fractional version of the k-Paging [9] problem and based on the optimal eviction policy (*evict the page appearing the furthest in the future*), we design an algorithm producing optimal solutions for the LP with values being multiple of $1/r$.

[1] By omitting the $\log n$-multiplication step of Step 7, one could establish that the moving cost of the produced permutations is at most 4 times the moving cost of the matrix-sequence, however omitting the $\log n$ multiplication could lead in prohibitively high covering cost.

[2] Step 3 of Algorithm 9 is well-defined since $|R_t| \leq r$ and $\sum_{e\in R_t} A^t_{e1} = 1$.

In the second step (Lemma 16), we show that for any sequence $\hat{A}^0 = \pi^0, \hat{A}^1, \ldots, \hat{A}^T$ satisfying properties **(i)** and **(ii)**, the moving cost of Algorithm 9 is at most $O(r^2) \cdot \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1})$. The latter is achieved through the use of an appropriate *potential function* based on a generalization of Kendall-Tau distance to doubly stochastic matrices with entries being multiples of $1/r$ (see Definition 18).

Theorem 10. *Algorithm 9 is a $O(r^2)$-approximation algorithm for* Mult-MSSC.

In Section 3.3 and 3.4 we provide the basic steps and ideas in the proof of Theorem 9 and 10 respectively.

3.3 Proof of Theorem 9

The basic step towards the proof of Theorem 9 is Lemma 13, establishing the fact that once two doubly stochastic matrices are given as input to the randomized rounding of Algorithm 8, the expected distance of the produced permutations is approximately bounded by the distance of the respective doubly stochastic matrices.

Lemma 13. *Let the doubly stochastic matrices A, B given as input to the rounding scheme of Algorithm 8. Then for the produced permutations π^A, π^B, $\mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi^A, \pi^B)\right] \leq 4 \log^2 n \cdot \mathrm{d}_{\mathrm{FR}}(A, B)$.*

Before exhibiting the proof of Lemma 13 we introduce the notion of *neighboring matrices*.

Definition 13. *(Neighboring stochastic matrices) The stochastic matrices A, B are neighboring if and only if they differ in exactly two entries lying on the same row and on consecutive columns.*

Let $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $B = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and $C = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. The pair of matrices (A, B) and (A, C) are neighboring while (B, C) are not.

Any doubly stochastic matrix A can be converted to another doubly stochastic matrix B through an intermediate sequence of neighboring stochastic matrices all of which are *almost doubly stochastic* and their overall moving cost equals $\mathrm{d}_{\mathrm{FR}}(A, B)$.

Claim 2. *Given the doubly stochastic matrices A, B, there exists a finite sequence of stochastic matrices, $A^0, \ldots, A^T$ such that*

1. *$A^0 = A$ and $A^T = B$.*
2. *A^t and A^{t-1} are neighboring.*
3. *the column-sum is bounded by 2, $\sum_{e \in U} A^t_{ei} \leq 2$ for all $1 \leq i \leq n$.*
4. *$\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1}) = \mathrm{d}_{\mathrm{FR}}(A, B)$.*

Proof Sketch of Claim 2. Let f^e_{ij} denotes the optimal solution of the linear program of Definition 10 defining the FootRule distance $\mathrm{d}_{\mathrm{FR}}(A, B)$. In case $A \neq B$, there exist elements e_1, e_2 and indices $i < j$ such that $f^{e_1}_{i\ell(i)} > 0$ and $f^{e_2}_{j\ell(j)} > 0$ with $\ell(i) >= j$ and $\ell(j) <= i$.

Let $\epsilon = \min(f^{e_1}_{i\ell(i)}, f^{e_2}_{j\ell(j)})$ and consider the sequence of the $|i - j|$ matrices produced by moving ϵ amount of mass in row e_1 from column i to column j. Then consider the sequence of the $|i - j|$ matrices produced by moving ϵ amount of mass in the row e_2 from column j to column i.

In the overall sequence of $2|i - j|$ stochastic matrices, two consecutive matrices are *neighboring*. Furthermore the column-sum of the matrices does not exceed $1 + \epsilon \leq 2$ and the final matrix A' of the sequence is doubly stochastic. Moreover by the fact that $t(i) \geq j$ and $t(j) \leq i$ we get that the overall moving cost of the sequence equals $\mathrm{d}_{\mathrm{FR}}(A, A')$ and that $\mathrm{d}_{\mathrm{FR}}(A, B) = \mathrm{d}_{\mathrm{FR}}(A, A') + \mathrm{d}_{\mathrm{FR}}(A', B)$. Applying the same argument inductively, until we reach matrix B, proves Claim 2. $\square$

Let the doubly stochastic matrices $A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \end{pmatrix}$. A can be converted to B with the following sequence neighboring stochastic matrices, $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 \end{pmatrix}$. Notice that the above sequence satisfies all the 4 requirements of Claim 2.

The notion of neighboring matrices is rather helpful since Lemma 13 admits a fairly simple proof in case A, B are neighboring stochastic matrices (notice that the rounding scheme of Algorithm 8 is well-defined even for stochastic matrices). The latter is formally stated and proven in Lemma 14 and is the main technical claim of the section.

Lemma 14. *Let π^A, π^B the permutations produced by the rounding scheme of Algorithm 8 (given as input) the stochastic matrices A, B that **i)** are neighboring **ii)** their column-sum is bounded by 2, then* $\mathbb{E}[\mathrm{d}_{\mathrm{KT}}(\pi^A, \pi^B)] \leq 4 \log^2 n \cdot \mathrm{d}_{\mathrm{FR}}(A, B)$

Proof of Lemma 14. Since A, B are neighboring there exists exactly two consecutive entries for which A, B differ, denoted as (e^*, i^*) and $(e^*, i^* + 1)$. Let $\epsilon := A_{e^* i^*} - B_{e^* i^*}$, by the Definition 10 of FootRule distance, we get that $\mathrm{d}_{\mathrm{FR}}(A, B) = |\epsilon|$. Without loss of generality we consider $\epsilon > 0$ (the case $\epsilon < 0$ symmetrically follows). We also denote with O_i the set of elements $O_i := \{e \neq e^\star \text{ such that } I^A_e = i\}$ and with I^A_e, I^B_e the indices in Step 6 of Algorithm 8.

Since A, B are neighboring, the e-th row of A and the e-th row of B are identical for all $e \neq e^\star$. As a result, $I^A_e = I^B_e$ for all $e \neq e^\star$. Furthermore the neighboring property implies that even for e^*, $\sum_{s=1}^{i} A_{e^\star s} = \sum_{s=1}^{i} B_{e^\star s}$ for all $i \neq i^\star$ and thus $\Pr\left[I^A_{e^\star} = i \wedge I^B_{e^\star} = j\right] = 0$ for $(i, j) \neq (i^\star, i^\star + 1)$. Now notice that

$$\begin{aligned}\Pr\left[I^A_{e^\star} = i^\star, I^B_{e^\star} = i^\star + 1\right] &\leq \Pr\left[\log n \cdot \sum_{s=1}^{i^\star} B_{e^\star s} \leq \alpha_e \leq \log n \cdot \sum_{s=1}^{i^\star} A_{e^\star s}\right] \\ &\leq \log n \cdot (A_{e^\star i^\star} - B_{e^\star i^\star}) = \log n \cdot \epsilon\end{aligned}$$

Notice also that in case $I^A_{e^\star} = I^B_{e^\star}$, $\mathrm{d}_{\mathrm{KT}}(\pi_A, \pi_B) = 0$. This is due to the fact that in such a case $I^A_e = I^B_e$ for all $e \in U$ and the fact that ties are broken lexicographically. As a result,

$$\begin{aligned}\mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi_A, \pi_B)\right] &= \Pr\left[I^A_{e^\star} \neq I^B_{e^\star}\right] \cdot \mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi_A, \pi_B) |\, I^A_{e^\star} \neq I^B_{e^\star}\right] \\ &= \Pr[I^A_{e^\star} = i^\star, I^B_{e^\star} = i^\star + 1] \cdot \mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi_A, \pi_B) |\, I^A_{e^\star} = i^\star, I^B_{e^\star} = i^\star + 1\right] \\ &\leq \epsilon \log n \cdot \left(\mathbb{E}\left[|O_{i^\star}|\right] + \mathbb{E}\left[|O_{i^\star+1}|\right]\right)\end{aligned}$$

where the last inequality follows by the fact that once $I^A_{e^*} = i^*$ and $I^B_{e^*} = i^* + 1$, the element e^* can move at most by $|O_{i^*}| + |O_{i^*+1}|$ positions and the fact that $I^A_{e^*}, I^B_{e^*}$ and $|O_{i^*}|, |O_{i^*+1}|$ are independent random variables.

We complete the proof we providing a bound on $\mathbb{E}\left[|O_i|\right]$. Notice that for $e \in U / \{e^*\}$,

$$\Pr[e \in O_i] \leq \Pr\left[\log n \sum_{s=1}^{i-1} A_{es} \leq \alpha_e \leq \log n \sum_{s=1}^{i} A_{es}\right] \leq \log n \cdot A_{ei}$$

which implies that $\mathbb{E}\left[|O_i|\right] \leq \log n \sum_{e \neq e^\star} A_{ei} \leq 2 \log n$. Finally we overall get,

$$\mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi_A, \pi_B)\right] \leq 4 \log^2 n \cdot \mathrm{d}_{\mathrm{FR}}(A, B)$$

□

The proof of Lemma 13 easily follows by Claim 2 and Lemma 14.

Proof of Lemma 13. Given the doubly stochastic matrices A, B, let the sequence $A = A^0, A^1, \dots, A^T = B$ of neighboring stochastic matrices ensured by Claim 2. Now let $\pi^0, \pi^1, \dots, \pi^T$ the sequence of permutations that the randomized rounding of Algorithm 8 produces given as input the sequence $A = A^0, A^1, \dots, A^T = B$. Notice that,

$$\mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi^A, \pi^B)\right] \leq \sum_{t=1}^{t} \mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})\right] \leq 4\log^2 n \cdot \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1}) = 4\log^2 n \cdot \mathrm{d}_{\mathrm{FR}}(A, B)$$

where the first inequality follows by the triangle inequality, the second by Lemma 14 and the last equality by Case 4 of Claim 2. ☐

We conclude the section with the proof of Theorem 9.

Proof of Theorem 9. By Lemma 13 and Lemma 12,

$$\sum_{t=1}^{T} \mathbb{E}\left[\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1})\right] \leq 4\log^2 n \cdot \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1}) \leq 4\log^2 n \cdot \mathrm{OPT}_{\text{Mult-MSSC}}$$

Up next we bound the expected covering cost $\sum_{t=1}^{T} \mathbb{E}\left[\pi^t(R_t)\right]$. Notice that since $\sum_{e \in R_t} A^t_{e1} = 1$, the only elements that can have index $I^t_e = 1$ are the elements $e \in R_t$. As a result, in case there exists some e at round t with $I^t_e = 1$ then $\pi^t(R_t) = 1$.

$$\begin{aligned}
\mathbb{E}\left[\pi^t(R_t)\right] &\leq 1 + n \cdot \Pr\left[I^t_e > 1 \text{ for all } e \in R_t\right] \\
&\leq 1 + n \cdot \Pi_{e \in R_t}\left(1 - \log n \cdot A^t_{e1}\right) \\
&\leq 1 + n \cdot e^{-\log n \cdot \sum_{e \in R_t} A^t_{e1}} = 2 \cdot \pi^t_{\mathrm{Opt}}(R_t)
\end{aligned}$$

where the last inequality follows due to the fact that $\sum_{e \in R_t} A^t_{e1} = 1$ and $\pi^t_{\mathrm{Opt}}(R_t) \geq 1$. ☐

3.4 Proof of Theorem 10

In this section we present the basic steps towards the proof of Theorem 10. We remind that $|R_t| \leq r$ and we denote with e_t the element that Algorithm 9 moves in the fist position at round t. As already mentioned, the proof is structured in two different steps.

1. We prove the existence of a sequence of doubly stochastic matrices $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ such that **(i)** the entries of each $\hat{A}^t$ are multiples of $1/r$, **(ii)** each $\hat{A}^t$ admits $1/r$ mass for element e_t in first position ($\hat{A}^t_{e_t 1} \geq 1/r$) and **(iii)** $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1}) \leq \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$.

2. We use properties **(i)** and **(ii)** to prove that the moving cost of Algorithm 9 is roughly upper bounded by $\Theta(r^2) \cdot \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1})$.

We start with the construction of the sequence $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$.

Definition 14. *For the sequence of elements $e_1, \dots, e_T \in U$ (the elements that Algorithm 9 moves to the fist position at each round), consider the following linear program,*

$$\begin{array}{lll}
\textit{min} & \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1}) & \\
\textit{s.t} & \sum_{i=1}^{n} \hat{A}^t_{ei} = 1 & \textit{for all } e \in U \textit{ and } t = 1, \dots, T \\
& \sum_{e \in U} \hat{A}^t_{ei} = 1 & \textit{for all } i = 1, \dots, n \textit{ and } t = 1, \dots, T \\
& \hat{A}^t_{e_t 1} \geq 1/r & \textit{for all } t = 1, \dots, T \\
& \hat{A}^0 = \pi^0 & \\
& \hat{A}^t_{ei} \geq 0 & \textit{for all } e \in U,\ i = 1, \dots, n \textit{ and } t = 1, \dots, T
\end{array}$$

The sequence $\hat{A}^0 = \pi^0, \dots, \hat{A}^T$ is defined as the optimal solution of the LP in Definition 14 with the entries of each $\hat{A}^t$ being **multiples of** $1/r$. The existence of such an optimal solution is established in Lemma 15.

Lemma 15. *There exists an optimal solution $\hat{A} = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ for the linear program of Definition 15 such that entries of each $\hat{A}^t$ are multiples of $1/r$.*

The proof of Lemma 15 is one of the main technical contributions of this work. Due to lack of space its proof is deferred to the full version of the paper. We remark that the *semi-integrality property*, that Lemma 15 states, is not due to the properties of the LP's polytope and in fact there are simple instances in which the optimal extreme points do not satisfy it. We establish Lemma 15 via the design of an optimal algorithm for the LP of Definition 14 (Algorithm 11) that always produces solutions with entries being multiples of $1/r$. Up next we describe in brief the idea behind Algorithm 11.

Given the matrix $\hat{A}^{t-1}$, Algorithm 11 construct $\hat{A}^t$ as follows. At first it moves $1/r$ mass from the left-most entry (e_t, j) with $\hat{A}^{t-1}_{e_t j} \geq 1/r$ to the entry $(e_t, 1)$. At this point the third constraint of the LP in Definition 14 is satisfied but the column-stochasticity constraints are violated (the first column admits mass $1 + 1/r$ and the j-th column admits mass $1 - 1/r$). Algorithm 11 inductively restores column-stochasticity from left to right. At step i, all the columns on the left of i are restored and the violations concern the column i and j (i's mass is $1 + 1/r$ and j's mass is $1 - 1/r$). Now Algorithm 11 must move a total of $1/r$ mass from column i to column $i + 1$. In case there exists an element e with total amount of mass greater than $2/r$, Algorithm 9 moves the $1/r$ mass from the entry (e, i) to the entry $(e, i + 1)$. The reason is that even if $e = e_{t'}$ at some future round t', the third constraint only requires $1/r$ mass. In case there is no such element, Algorithm 11 moves the $1/r$ mass from the element appearing the furthest in the sequence $\{e_t, \dots, e_T\}$. The latter is in accordance with the optimal eviction policy for $\mathrm{k} - \mathrm{Paging}$ which at each round evicts the page appearing furthest in the future [9]. The optimality of Algorithm 11 is established in Lemma 17 and the fact that produced solution admits values being $1/r$ is inductively established.

To this end, we can show that all of the desired properties of the sequence $\hat{A} = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ are satisfied. Property **(i)** is established by Lemma 15. Property **(ii)** is enforced by the constraint $\hat{A}^t_{e_t 1} \geq 1/r$. Now for Property **(iii)**, notice that by the definition of Algorithm 9, $A^t_{e_t 1} \geq 1/r$. As a result, the sequence $A^0 = \pi^0, A^1, \dots, A^T$ is feasible for the linear program of Definition 14 and thus $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1}) \leq \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(A^t, A^{t-1})$.

Lemma 16. *Let $\pi^0, \pi^1, \dots, \pi^T$ the permutations produced by Algorithm 9 and $e_1, \dots, e_T$ the elements that Algorithm 9 moves to the first position at each round t. For any sequence of doubly stochastic matrices $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ for which Property **(i)** and Property **(ii)** are satisfied, $\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1}) \leq 2r^2 \cdot \sum_{t=1}^{T} \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1}) + r \cdot T$.*

The proof of Theorem 10 directly follows by Lemma 15 and 16. In Section 3.4.2 we present the basic steps for of Lemma 15.

3.4.1 Proof of Lemma 15

We prove the existence of an optimal solution $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ for the linear program of Definition 14 for which the entries of each matrix $\hat{A}^t$ are multiples of $1/r$ though the design of an optimal greedy algorithm illustrated in Algorithm 11.

The fact that Algorithm 11 produces a solution with entries that multiples of $1/r$ easily follows. Algorithm 11 starts with an integral doubly stochastic matrices ($\hat{A}^0 = \pi^0$) and always moves $1/r$ mass from entry to entry. The optimality of Algorithm 11 is established in Lemma 17 the proof of which is presented in the next section since it is quite technically complicated. However the basic idea of the algorithms is very intuitive, once $\hat{A}^{t-1}_{e_t} = 0$ Algorithm 11 moves $1/r$ mass of e_t from its leftmost position (with mass greaer than $1/r$), denoted as Pos of Step 5. At this point of time, Algorithm 11 has violated the column-stochasticity constraints, $1 + 1/r$ for the first column and $1 - 1/r$ for the Pos-th column and Algorithm 11 must move at total of $1/r$ mass from the first position to next positions until $1/r$ mass reaches the Pos position and column-stochasticity is restored (Step 8). Once Algorithm 11 detects an element with aggregated mass (until position j) $\geq 2/r$, it can safely move $1/r$ of each mass to position $j + 1$ since even if this element appears at some point in the future only $1/r$ is necessary to satisfy the constraint $A^t_{e_t 1} \geq 1/r$ and thus the rest is redundant (Step 11). In case such an element does not exist, Algorithm 11 moves the (useful) $1/r$ mass of the element appearing the furthest in the remaining sequence $\{e_t, \dots, e_T\}$, which is exactly the same optimal *eviction policy* that the well-studied $k-$ Paging suggests.

Lemma 17. *Algorithm 11 produces an optimal solution $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ for the linear program of Definition 14 while the entries of each $\hat{A}^t$ are multiples of $1/r$.*

Proof of Lemma 17

At first notice that the linear program of Definition 10 defining the FootRule distance $\mathrm{d_{FR}}(A, B)$ between the stochastic matrices A, B can take the following equivalent form.

$$\begin{aligned}
\min \quad & \sum_{e\in U}\sum_{i=1}^{n} (P_{ei} + M_{ei}) \\
\text{s.t} \quad & P_{ei} - M_{ei} = \sum_{s=1}^{i} (A_{es} - B_{es}) \quad \text{for all } e \in U \text{ and } i = 1, \dots, n \\
& P_{ei}, M_{ei} \geq 0 \qquad\qquad\qquad\quad \text{for all } e \in U \text{ and } i = 1, \dots, n
\end{aligned}$$

This is due to the fact that $(P_{ei} + M_{ei})$ takes the value $|\sum_{s=1}^{i}(A_{es} - B_{es})|$ and it is not hard to prove that $\sum_{e\in U} |\sum_{i=1}^{n}(A_{es} - B_{es})|$ equals $\mathrm{d_{FR}}(A, B)$. As a result, the linear program of Definition 14 takes the following equivalent form,

$$\begin{aligned}
\min \quad & \sum_{t=1}^{T}\sum_{e=1}^{n}\sum_{i=1}^{n} \left(P^t_{ei} + M^t_{ei}\right) \\
\text{s.t} \quad & P^t_{ei} - M^t_{ei} = \sum_{s=1}^{i} \left(\hat{A}^t_{es} - \hat{A}^{t-1}_{es}\right) \quad \text{for all } e \in U \text{ and } t \in \{1, T\} \\
& \sum_{e\in U} \hat{A}^t_{ei} = 1 \qquad\qquad \text{for all } t \in \{1, T\} \text{ and } i \in \{1, n\} \\
& \sum_{i=1}^{n} \hat{A}^t_{ei} = 1 \qquad\qquad \text{for all } t \in \{1, T\} \text{ and } e \in U \\
& \hat{A}^t_{e_t 1} \geq 1/r \qquad\qquad \text{for all } t \in \{1, T\} \\
& \hat{A}_0 = \pi_0 \\
& \hat{A}^t_{ei}, P^t_{ei}, M^t_{ei} \geq 0
\end{aligned}$$

In Definition 15 we construct n different linear programs admitting the property that the sum of their optimal values acts as a lower bound on the optimal solution of the linear program of Definition 14 and will help us establish the optimality of Algorithm 11.

Definition 15. *For each* $1 \leq i \leq n$ *consider the following linear program,*

$$\begin{array}{lll}
\textit{min} & \sum_{t=1}^{T}\sum_{e=1}^{n}\left(X_{ei}^{t}+Y_{ei}^{t}\right) & \\
\textit{s.t} & X_{ei}^{t}-Y_{ei}^{t}=B_{ei}^{t}-B_{ei}^{t-1} & \textit{for all } e\in U \textit{ and } t\in\{1,T\} \\
 & \sum_{e\in U}B_{ei}^{t}=i & \textit{for all } t\in\{1,T\} \\
 & B_{e_t i}^{t}\geq 1/r & \textit{for all } t\in\{1,T\} \\
 & B_{ei}^{0}=\sum_{s=1}^{i}\hat{A}_{es}^{0} & \textit{for all } e\in U \\
 & X_{ei}^{t},Y_{ei}^{t},B_{ei}^{t}\geq 0 & \textit{for all } e\in U \textit{ and } t\in\{1,T\}
\end{array}$$

Let $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ denote the optimal solution of the linear program of Definition 14. Notice that,

$$\sum_{t=1}^{T}\mathrm{d}_{\mathrm{FR}}(\hat{A}^t,\hat{A}^{t-1}) \geq \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{e\in U}\left(X_{ei}^{t}+Y_{ei}^{t}\right)$$

where X_{ei}^t, Y_{ei}^t denote the values of the respective variables in the optimal solution of the i-th linear program in Definition 15. This is due to the fact that setting $B_{ei}^t = \sum_{s=1}^{i} A_{es}^t$ produces a feasible solution for the i-th linear program in Definition 15. We will prove that for the sequence $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ produced by Algorithm 11,

$$\sum_{t=1}^{T}\mathrm{d}_{\mathrm{FR}}(\hat{A}^t,\hat{A}^{t-1}) = \sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{e\in U}\left(X_{ei}^{t}+Y_{ei}^{t}\right)$$

which implies optimality.
To this end, we present a very natural interpretation of the i-th linear program in Definition 15 that will help us design an optimal greedy algorithm (Algorithm 10) for solving the linear programs of Definition 15. Each of the linear programs of Definition 15 can be viewed as a *fractional version* of the well-known $k-$ Paging []. In Definition 16 we provide this alternative and more intuitive definition for the linear programs of Definition 15.

Definition 16 (Fractional Paging)**.** *Given a sequence of elements* $e_1, \dots, e_T$ *and an initial vector* S^0 *such that* $|S^0| = n$, $0 \leq S_e^0 \leq 1$ *and* $\sum_{e\in U} S_e^0 = 1$. *Compute a sequence of vectors* $S_1, \dots, S_T$ *such that*

1. $0 \leq S_e^t \leq 1$
2. $\sum_{e\in U} S_e^t = 1$
3. $S_{e_t}^t \geq 1/r$ *for* $1 \leq t \leq T$

and the quantity $\sum_{t=1}^{T}\sum_{e\in U}|S_e^t - S_e^{t-1}|$ *is minimized.*

In Algorithm 10 we present a generalization the classical greedy eviction policy (*evict the page arriving latter in the future*) that is the optimal policy for the original paging problem.

Algorithm 10 Greedy Algorithm for Fractional Paging

Input: An initial vector S_0 and a sequence of elements $e_1, \dots, e_T \in U$

Output: A sequence of vectors $S_1, \dots, S_T$.

```
1: for $t = 1$ to $T$ do
2:   $S^t \leftarrow S^{t-1}$
3:   $S^t_{e_t} \leftarrow S^t_{e_t} + \min\left(1/r - S^t_{e_t}\right)$
4:   for each element $e \in U$ do
5:     //Remove first the redundant mass.
6:     if $S^t_e \geq 2/r$ and $\sum_{e \in U} S^t_e \geq i$ then
7:       $S^t_e \leftarrow S^t_e - \min\left(S^t_{e_t} - 1/r, \sum_{e \in U} S^t_e - i\right)$
8:     end if
9:     //If redundant mass is not enough, remove mass from the elements arriving latter in the future.
10:    $t^\star(e) \leftarrow$ the first time $s \geq t$ such that $e_s = e$.
11:    Sort the elements in decreasing order according to $t^\star(e)$.
12:    for all elements $e \in U$ (according to the previous ordering) do
13:      $S^t_e \leftarrow S^t_e - \min(S^t_e, \sum_{e \in U} S^t_e - i)$.
14:    end for
15:  end for
16: end for
17: return $S_1, \dots, S_T$
```

Lemma 18. *Algorithm 10 is optimal for the fractional paging.*

Proof. Let O^t denote the vector of the optimal solution of the Fractional Paging of Definition 16 at round t, for $1 \leq t \leq T$. Without loss of generality we assume that $O^t_\alpha \geq O^{t-1}_\alpha$ for $\alpha = e_t$ and $O^t_\alpha \leq O^{t-1}_\alpha$ for $\alpha \neq e_t$.

We first construct a sequence of vectors S^t for $1 \leq t \leq T$ such that the vector S^1 agrees with Algorithm 10 and the cost of the vector sequence $S^1, \dots, S^T$ is optimal. Once this established the proof follows inductively.

Before presenting the construction we partition the elements into the following 3 classes. The set *neutral* denoted by N_t denotes the elements e for which $S^t_e = \min(1/r, O^t_e)$. The set of *greater elements at round* t, denoted by G_t, which are all elements $e \notin N_t$ and $S^t_e \geq O^t_e$. Finally we have the set of *smaller elements* L_t which are all $e \notin N_t$ and $S^t_e < O^t_e$. The vector sequence $\{S^t\}_{2 \leq t \leq T}$ is inductively defined as follows: **For each round** $t \geq 2$**,**

1. If $O^t_e \geq O^{t-1}_e$
   - If $e \in N_{t-1}$ then $S^t_e = \min(1/r, O^t_e)$.
   - If $e \in G_{t-1}$ then $S^t_e = S^{t-1}_e + \min\left(\min(O^t_e, 1/r) - S^{t-1}_e, 0\right)$.
   - If $e \in L_{t-1}$ then $S^t_e = S^{t-1}_e + \min(O^t_e - O^{t-1}_e, 1/r - S^{t-1}_e)$.
2. If $O^t_e < O^{t-1}_e$
   - If $e \in N^{t-1}$ then $S^t_e = \min(1/r, O^t_e)$.
   - If $e \in G^{t-1}$ then $S^t_e = S^{t-1}_e - O^{t-1}_e + O^t_e$.
   - If $e \in L^{t-1}$ then $S^t_e = S^{t-1}_e - \max(S^{t-1}_e - O^t_e, 0)$.

Finally in case $\sum_{e \in U} S^t_e > i$, we additionally subtract total amount of $\sum_{e \in U} S^t_e - i$ from the elements $S^t_e \geq O^t_e$. As a result, the cost of round t is

$$\sum_{e \in S^t_e \geq S^{t-1}_e} (S^t_e - S^{t-1}_e) + \sum_{e \in S^t_e \leq S^{t-1}_e} (S^{t-1}_e - S^t_e) + \sum_{e \in U} S^t_e - i = 2 \sum_{e \in S^t_e \geq S^{t-1}_e} (S^t_e - S^{t-1}_e)$$

By the definition of S^t,

$$2\sum_{e\in S_e^t \geq S_e^{t-1}} (S_e^t - S_e^{t-1}) = 2\sum_{e\in O_e^t \geq O_e^{t-1}} (S_e^t - S_e^{t-1}) \leq 2\sum_{e\in O_e^t \geq O_e^{t-1}} (O_e^t - O_e^{t-1}) = ||O^t - O^{t-1}||_1$$

As a result, we overall get that $\sum_{t=1}^T ||S^t - S^{t-1}||_1 \leq \sum_{t=1}^T ||O^t - O^{t-1}||_1$.

Up next we prove that the solution $S_1, \dots, S_T$ is a feasible solution for *fractional paging*.

At first observe that an element e can only go from the state form the state of *greater* to the state of *neutral* and from the state of *smaller* to the state of *neutral*. Moreover observe that *once an element becomes neutral it remains neutral forever*.

The only case that the constructed solution $S^1, \dots, S^T$ is not feasible is by having $e \in L^t$ with $e = e_t$ for some round t (otherwise $S_e^t \geq O_e^t \geq 1/r$). Combining the latter with the previous observation we get that $e \in L^\ell$ for all $1 \leq \ell \leq t$. Moreover $S_e^t < 1/r$.

We consider the mutually exclusive cases $S_e^1 < 1/r$ and $S_e^1 \geq 1/r$.

Let us start with $S_e^1 < 1/r$. By Algorithm 10, we get that $e \neq e_1$ and thus $O_e^1 \leq S_e^0$ ($S^0 = O^0$) and since $e \in L^1$, we get that $S_e^1 < S_e^0$. Since $S_e^1 < 1/r$ and $S_e^1 < S_e^0$ by Steps $4-8$ of Algorithm 10, we get that for $\alpha \in U$, $S_\alpha^1 \leq 1/r$ (all the redundant mass is removed before useful is removed). Moreover by Steps $11-14$ we get that for all $\alpha \in G^1$, $t^*(\alpha) < t^*(e)$[3]. Finally notice that in case $S_\alpha^{t-1} < 1/r, \alpha = e_t$ and $\alpha \in G^{t-1}$ then α becomes neutral, $\alpha \in N^t$. This implies that until round $t^*(e)$ all elements $\alpha \in G^1$ have become neutral. As a result, at round $t^\star(e)$ all elements $\alpha \in G^1$ have become neutral which means that for all elements $\alpha \in U/\{e\}$, $S_\alpha^t \leq O_\alpha^t$. If $e \in L^t$ then $\sum_{\alpha\in U} S_\alpha^t < \sum_{\alpha\in U} O_\alpha^t$ which is a contradiction. Thus $S_e^{t^*(e)} \geq 1/r$.

In either the case $S_e^1 \geq 1/r$ or $S_e^{t^*(e)} \geq 1/r$ (that follows in case $S_e^1 < 1/r$) we get that there exists an $\ell \leq t$ such that $S_e^{\ell-1} \geq 1/r$ and $S_e^\ell < 1/r$. Since $e \in L^{\ell-1}$, we get that $S_e^\ell = S_e^{\ell-1} - \max(S_e^{\ell-1} - O_e^\ell, 0)$ which implies that $S_e^\ell = O_e^\ell$ and thus e becomes neutral. $\square$

We conclude the section with the proof of Lemma 17. We set $B_{ei}^t = \sum_{s=1}^i \hat{A}_{es}^t$ where $\hat{A}^0 = \pi^0, \hat{A}^1, \dots, \hat{A}^T$ are the stochastic matrices produced by Algorithm 11. We show that for $i = 1, \dots, n$ the constructed sequence $B_{ei}^0, \dots, B_{ei}^T$ is *consistent* with Algorithm 10, meaning that the $B_{ei}^0, \dots, B_{ei}^T$ could have been the output of Algorithm 10. As a result,

1. Each $\sum_{t=1}^T \sum_{e\in U} \left(X_{ei}^t + Y_{ei}^t\right)$ equals the optimal value of the i-th linear program in Definition 15.
2. $\sum_{i=1}^n \sum_{t=1}^T \sum_{e\in U} \left(X_{ei}^t + Y_{ei}^t\right) = \sum_{t=1}^T \mathrm{d}_{\mathrm{FR}}(\hat{A}^t, \hat{A}^{t-1})$ since we can set $P_{ei}^t = X_{ei}^t$ and $M_{ei}^t = Y_{ei}^t$ ($B_{ei}^t = \sum_{s=1}^i \hat{A}_{es}^t$).

The latter two properties establish the optimality of Algorithm 11.
We now prove that the sequence $B_{ei}^0, \dots, B_{ei}^T$ where $B_{ei}^t = \sum_{s=1}^i \hat{A}_{es}^t$ is consistent with Algorithm 10. We emphasize that we prove this under the condition that the initial vector B_{ei}^0 is a $0-1$ vector which simplifies a lot the actions of Algorithm 10. In particular and an easy induction argument reveals that when the initial vector B_{ei}^0 is a $0-1$ vector then the vectors produced by Algorithm 10 admit entries with multiples of $1/r$, something that is obviously true for the sequence $B_{ei}^0, \dots, B_{ei}^T$ since $B_{ei}^t = \sum_{s=1}^i \hat{A}_{es}^t$.

[3] Let $\alpha \in G_1$ with $t^*(\alpha) > t^*(e)$ then by Step 11 and 12 of Algorithm 10 $S_\alpha^1 = 0$, something that cannot be true, since $\alpha \in G_1$.

Algorithm 11 An Optimal Greedy Algorithm for the LP of Definition 14

Input: The initial permutation π^0 and the sequence of elements $e_1, \dots, e_T \in U$

Output: An optimal solution of a linear program of Definition 14 where the entries of $\hat{A}^t$ are multiples of $1/r$.

```
1: Initially $\hat{A}^0 \leftarrow \pi_0$
2: for all rounds $t = 1$ to $T$ do
3:   $\hat{A}^t \leftarrow \hat{A}^{t-1}$
4:   if $\hat{A}^t_{e_t 1} < 1/r$ then
5:     //Move $1/r$ mass of $e_t$ to the first position
6:         Pos $\leftarrow$ argmin$_{1\leq i \leq n}\{A^t_{ei} \geq 1/r\}$
7:         $\hat{A}^t_{e1} \leftarrow \hat{A}^t_{e1} + 1/r, \hat{A}^t_{e\text{Pos}} \leftarrow \hat{A}^t_{e\text{Pos}} - 1/r$
8:     //Restore the column-stochasticity constraints from left to right
9:     for $j = 1$ to Pos $- 1$ do
10:      if there exists $e \in U$ with $\sum_{s=1}^{j} \hat{A}^t_{es} \geq 2/r$ and $\hat{A}^t_{es} \geq 1/r$ then
11:        //Move $1/r$ of its (redundant) mass to the next position
12:          $\hat{A}^t_{ej} \leftarrow \hat{A}^t_{ej} - 1/r, \hat{A}^t_{ej} \leftarrow \hat{A}^t_{ej} + 1/r$
13:      else
14:        //Move the $1/r$ mass, of the element appearing furthest in the future, to the next position
15:            $e^\star \in U \leftarrow$ the element with $\hat{A}^t_{e^\star j} = 1/r$ furthest in $\{e_{t+1}, \dots, e_T\}$
16:            $\hat{A}^t_{e^\star j} \leftarrow \hat{A}^t_{e^\star j} - 1/r, \hat{A}^t_{e^\star j} \leftarrow \hat{A}^t_{e^\star j} + 1/r$
17:      end if
18:    end for
19:  end if
20: end for
21: return $\hat{A}_1, \dots, \hat{A}_T$
```

Let $B^t_{e_t i} = B^{t-1}_{e_t i}$, this is the case where $B^{t-1}_{e_t i} \geq 1/r$. In order to ensure consistency with Algorithm 10, we need to show that $B^t_{ei} = B^{t-1}_{ei}$ for all $e \in U$ (see Step 3 of Algorithm 10). By the fact that $B^t_{e_t i} = B^{t-1}_{e_t i}$ we get that $\sum_{s=1}^{i} \hat{A}^t_{e_t s} = \sum_{s=1}^{i} \hat{A}^{t-1}_{e_t s}$ which by Algorithm 11 implies that $\hat{A}^{t-1}_{ej} \geq 1/r$ for some $j \leq i$. The latter together with Algorithm 11 (see Step 9 of Algorithm 11) implies that $\sum_{s=1}^{i} \hat{A}^t_{ei} = \sum_{s=1}^{i} \hat{A}^{t-1}_{ei}$ for all $e \in U$. Thus, $B^t_{ei} = B^{t-1}_{ei}$.

Let $B^t_{e_t i} = B^{t-1}_{e_t i} + 1/r$, this is the case where $B^{t-1}_{e_t i} = 0$. To establish consistency with Algorithm 10 we need to prove that there exists a unique e^* with $B^t_{e^* i} = B^{t-1}_{e^* i} - 1/r$ and one of the following holds,

1. $B^{t-1}_{e^* i} \geq 2/r$ (Condition 6 in Algorithm 10)

2. $B^{t-1}_{ei} \leq 1/r$ for all $e \in U$ and $B^{t-1}_{e^* i} = 1/r$ is the element appearing furthest in the sequence $\{e_{t+1}, \dots, e_T\}$ (Condition 6 in not met and Algorithm 10 continues in Steps $11 - 14$).

Step $9 - 14$ of Algorithm 11 guarantees that the existence of a unique element e^* such that $\sum_{s=1}^{i} \hat{A}^t_{e^* s} = \sum_{s=1}^{i} \hat{A}^{t-1}_{e^* s} - 1/r$ which implies the existence of a unique element e^* with $B^t_{e^* i} = B^{t-1}_{e^* i} - 1/r$ since $B^t_{e^* i} = \sum_{s=1}^{i} \hat{A}^t_{e^* s}$. In case $\sum_{s=1}^{i} \hat{A}^{t-1}_{e^* s} \geq 2/r$ then we are done. So let us assume that $\sum_{s=1}^{i} \hat{A}^{t-1}_{e^* s} = 1/r$. This implies that $\sum_{s=1}^{i} \hat{A}^{t-1}_{es} \leq 1/r$ for all $e \in U$ since otherwise Algorithm 11 would have moved an element e' with $\sum_{s=1}^{i} \hat{A}^{t-1}_{e' s} \geq 2/r$ (see Step 11 of Algorithm 11). The fact that e^* is the element with $\sum_{s=1}^{i} \hat{A}^{t-1}_{e^* s} = 1/r$ appearing furthest in the sequence $\{e_{t+1}, \dots, e_T\}$ is ensured by Step 14 of Algorithm 11.

3.4.2 Proof of Lemma 16

In order to prove Lemma 16, we make use of an appropriate potential function that can be viewed as an extension of the Kendall-Tau distance (see Definition 7) to doubly stochastic matrices with entries being multiples of $1/r$.

Definition 17 (r-Index). *The r-index of an element $e \in U$ in the doubly stochastic matrix A, $I_e^A :=$* $\text{argmin}\{1 \leq i \leq n : \sum_{s=1}^{i} A_{es} \geq 1/r\}$

Definition 18 (Fractional Kendall-Tau Distance). *Given the doubly stochastic matrices A, B, a pair of elements $(e, e') \in U \times U$ is inverted if and only if one of the following condition holds,*

1. *$I_e^A > I_{e'}^A$ and $I_e^B < I_{e'}^B$.*
2. *$I_e^A < I_{e'}^A$ and $I_e^B > I_{e'}^B$.*
3. *$I_e^A = I_{e'}^A$ and $I_e^B \neq I_{e'}^B$.*
4. *$I_e^A \neq I_{e'}^A$ and $I_e^B = I_{e'}^B$.*

The fractional Kendall-Tau distance between two doubly stochastic matrices A, B, denoted as $\text{d}_{\text{KT}}(A, B)$*, is the number of inverted pairs of elements.*

Notice that in case of $0 - 1$ doubly stochastic matrices the Fractional Kendall-Tau distance of Definition 18 coincides with the Kendall-Tau distance of Definition 7.

Claim 3. *Fractional Kendall-Tau Distance satisfies the triangle inequality,* $\text{d}_{\text{KT}}(A, B) \leq \text{d}_{\text{KT}}(A, C) + \text{d}_{\text{KT}}(C, B)$.

Proof of Claim 3. Let $X_{ee'}^{AB} = 1$ if (e, e') is inverted pair for the matrices A, B and 0 otherwise (respectively for $X_{ee'}^{AC}, X_{ee'}^{BC}$). By a short case study one can show that once $X_{ee'}^{AB} = 1$ then $X_{ee'}^{AC} + X_{ee'}^{BC} \geq 1$ which directly implies Claim 3. □

In the case of doubly stochastic matrices with their entries being multiples of $1/r$, Fractional Kendall-Tau distance relates to FootRule distance of Definition 10.

Lemma 19. *Let the doubly stochastic matrices A, B with entries that are multiples of $1/r$. Then* $\text{d}_{\text{KT}}(A, B) \leq 2r^2 \cdot \text{d}_{\text{FR}}(A, B)$.

Proof of Lemma 19. We construct a doubly stochastic matrix A' for which the following properties hold,

1. The entries of A' are multiples of $\frac{1}{r}$.
2. $\text{d}_{\text{FR}}(A, B) = \text{d}_{\text{FR}}(A, A') + \text{d}_{\text{FR}}(A', B)$.
3. $\text{d}_{\text{KT}}(A, A') \leq 2r^2 \cdot \text{d}_{\text{FR}}(A, A')$.

Once the above properties are established, Lemma 19 follows by repeating the same construction until matrix B is reached and by using the fact that the *fractional Kendall-Tau distance* of Definition 18 satisfies the triangle inequality.

Before proceeding with the construction of A', we present the following corollary that follows by an easy exchange argument.

Corollary 1. *Let the stochastic matrices A, B with entries multiples of $1/r$, the values f_{ij}^e of the optimal solution in the linear program of Definition 10 (the min-cost transportation problem defining the FootRule distance* $\text{d}_{\text{FR}}(A, B)$*) are multiples of $1/r$.*

In order to construct the matrix A' satisfying the Properties 1-3, we consider three different classes of the entries (e, i). In particular, we call an entry (e, i).

1. *right* if and only if $f^e_{ij} > 0$ for some $j > i$.
2. *left* if and only if $f^e_{ij} > 0$ for some $j < i$.
3. *neutral* if and only if $f^e_{ij} = 0$ for all $j \neq i$.

Note that the above classes do not form a partition of the entries since an entry (e, i) can be both *left* and *right* at the same time.

Corollary 2. *Given two doubly stochastic matrices $A \neq B$, there exist entries (e, i) and (e', j) such that*

1. $j > i$

2. the entry (e, i) is right

3. the entry (e', j) is left

4. the entry (α, ℓ) is neutral for all $\alpha \in U$ and $\ell \in \{i + 1, j - 1\}$

We construct the matrix A' from matrix A as follows. Consider two entries (e, i) and (e', j) with the properties that Corollary 2 illustrates. The doubly stochastic matrix A' is constructed by moving $1/r$ mass from entry (e, i) to entry (e, j) and by moving $1/r$ mass from entry (e', j) to entry (e', i). More formally,

$$A'_{\alpha\ell} = \begin{cases} A_{\alpha\ell} - \frac{1}{r} & \text{if } (\alpha, \ell) = (e, i) \\ A_{\alpha\ell} - \frac{1}{r} & \text{if } (\alpha, \ell) = (e', j) \\ A_{\alpha k} + \frac{1}{r} & \text{if } (\alpha, \ell) = (e', i) \\ A_{\alpha\ell} + \frac{1}{r} & \text{if } (\alpha, \ell) = (e, j) \\ A_{\alpha\ell} & \text{otherwise} \end{cases}$$

Up next we establish the fact that $\mathrm{d_{FR}}(A, B) = \mathrm{d_{FR}}(A, A') + \mathrm{d_{FR}}(A', B)$.

Claim 4. $\mathrm{d_{FR}}(A', A) = 2|j - i|/r$ *and* $\mathrm{d_{FR}}(A', B) = \mathrm{d_{FR}}(A, B) - 2|j - i|/r$.

Proof. The fact that $\mathrm{d_{FR}}(A', A) = 2|j - i|/r$ is trivial. We thus focus on showing that $\mathrm{d_{FR}}(A', B) = \mathrm{d_{FR}}(A, B) - 2|j - i|/r$.

Since (e, i) is *right*, there exists an index $\ell(i) > i$ such that $f^e_{i\ell(i)} > 0$. Moreover $f^e_{i\ell(i)} \geq 1/r$ since $f^e_{i\ell(i)}$ is multiple of $1/r$. Notice that $\ell(i) \neq \ell$ for $\ell \in \{i + 1, j - 1\}$ since all the entries (α, ℓ) are *neutral* (otherwise $\sum_{\alpha \in U} B_{\alpha\ell} > 1$). As a result, transfering $1/r$ mass from entry (e, i) to entry (e, j) decreases the FootRule distance between A and B by $1/r \cdot |i - j|$ since the *final destination* of the $1/r$ mass is the entry $(e, \ell(i))$ that is on the right of entry (e, j), $\ell(i) \geq j$. The claim follows by applying the exact same argument for (e', j). □

We now establish the last property that is $\mathrm{d_{KT}}(A, A') \leq 2r^2 \cdot \mathrm{d_{FR}}(A, A')$.

Claim 5. $\mathrm{d_{KT}}(A', B) \leq 4r \cdot |i - j|$

Proof. Notice that apart from e, e', the r-index of each element is the same in both A and A' ($I^A_\alpha = I^{A'}_\alpha$ for all $\alpha \in U \setminus \{e, e'\}$). As a result, by Definition 18, we get that the only inverted pairs can be of the form (e, α) or (e', α).

In case $I^A_e \leq i - 1$ then $I^A_e = I^{A'}_e$ and there is no inverted pair of the form (e, α). In case $I^A_e = i$ then $i \leq I^{A'}_e \leq j$ and any element α with $I^A_\alpha = I^{A'}_\alpha \in \{1, i - 1\} \cup \{j + 1, n\}$ cannot form an inverted pair with e. As a result, a pair (e, α) can be inverted only if $i \leq I_\alpha^A = I_\alpha^{A'} \leq j$. Since the entries of A are multiples of $1/r$ and A is doubly stochastic, there are at most r positive entries at each column of A. As a result, there are at most $r \cdot (j - i + 1)$ inverted pairs of the form (e, α). With the symmetric argument one can show that there are at most $r \cdot |j - i + 1|$ of the form (e', α). Overall there are at most $2r \cdot |j - i + 1|$ inverted pairs between A and A' that are less than $4r \cdot |j - i|$ since $j > i$. □

□

We conclude the section with Lemma 20. Then Lemma 16 follows by Lemma 20 and 19.

Lemma 20. *Let $\pi^0, \pi^1, \ldots, \pi^T$ the permutations produced by Algorithm 9 and $e_1, \ldots, e_T$ the elements that Algorithm 9 moves to the first position at each round t. For any sequence of doubly stochastic matrices $B^0 = \pi^0, B^1, \ldots, B^T$ with $B_{e_t 1}^t \geq 1/r$,*

$$\sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1}) \leq \sum_{t=1}^{T} \mathrm{d}_{\mathrm{KT}}(B^t, B^{t-1}) + r \cdot T$$

The proof of Lemma 20 is based on the following two inequalities, $\mathrm{d}_{\mathrm{KT}}(\pi^t, \pi^{t-1}) + \mathrm{d}_{\mathrm{KT}}(\pi^t, B^t) - \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, B^t) \leq r$ and $\mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, B^t) - \mathrm{d}_{\mathrm{KT}}(\pi^{t-1}, B^{t-1}) \leq \mathrm{d}_{\mathrm{KT}}(B^t, B^{t-1})$. The second inequality follows by the triangle inequality established in Claim 3. The first follows by the fact that $I_{e_t}^{B^t} = 1$ and the definition of Fractional Kendall-Tau distance.

Proof of Lemma 20. Since $B_{e_t}^t \geq 1/r$, the r-index of element e_t in matrix B^t is 1, $I_{e_t}^{B^t} = 1$. We first show that,

$$\mathrm{d}_{\mathrm{KT}}\left(\pi^t, \pi^{t-1}\right) + \mathrm{d}_{\mathrm{KT}}\left(\pi^t, B^t\right) - \mathrm{d}_{\mathrm{KT}}\left(\pi^{t-1}, B^t\right) \leq r$$

To simplify notation let k_t the position of e_t in π^{t-1}. Notice that $\mathrm{d}_{\mathrm{KT}}\left(\pi^t, \pi^{t-1}\right) = k_t - 1$. Out of the $k_t - 1$ elements lying on the left of e_t in π^{t-1} there are most $r - 1$ elements α with $I_\alpha^{B^t} = 1$ (these elements must admit $B_{\alpha 1}^t \geq 1/r$). The rest of the $k_t - 1$ elements admit r-index $I_\alpha^{B^t} \geq 2$ and thus form inverted pairs with e_t when considering π^{t-1} and B^t. When e_t moves to the first positions (permutation π^t) these inverted pairs are deactivated ($I_{e_t}^{B^t} = 1$) and new inverted pairs are created between e_t and α with $I_\alpha^{B^t} = 1$, but these new inverted pairs are at most r (for any element α with $I_\alpha^{B^t}$, $B_\alpha^t \geq 1/r$). Also notice no additional inverted pairs (e, α) (with $e \neq e_t$) are created since the order between all the other elements is the same in π^t and π^{t-1}. Overall,

$$\underbrace{\mathrm{d}_{\mathrm{KT}}\left(\pi^t, \pi^{t-1}\right)}_{k_t - 1} + \underbrace{\mathrm{d}_{\mathrm{KT}}\left(\pi^t, B^t\right) - \mathrm{d}_{\mathrm{KT}}\left(\pi^{t-1}, B^t\right)}_{\leq -k_t + 1 + r} \leq r$$

Combining the above inequality with $\mathrm{d}_{\mathrm{KT}}\left(\pi^{t-1}, B^t\right) - \mathrm{d}_{\mathrm{KT}}\left(\pi^{t-1}, B^{t-1}\right) \leq \mathrm{d}_{\mathrm{KT}}\left(B^t, B^{t-1}\right)$ which follows from the triangle inequality we get,

$$\mathrm{d}_{\mathrm{KT}}\left(\pi^t, \pi^{t-1}\right) + \mathrm{d}_{\mathrm{KT}}\left(\pi^t, B^t\right) - \mathrm{d}_{\mathrm{KT}}\left(\pi^{t-1}, B^{t-1}\right) \leq \mathrm{d}_{\mathrm{KT}}\left(A^t, B^{t-1}\right) + r.$$

Finally a telescopic sum gives $\sum_{t=1}^T \mathrm{d}_{\mathrm{KT}}\left(\pi^t, \pi^{t-1}\right) \leq \sum_{t=1}^T \mathrm{d}_{\mathrm{KT}}\left(B^t, B^{t-1}\right) + r \cdot T + \mathrm{d}_{\mathrm{KT}}(\pi^0, B^0) - \mathrm{d}_{\mathrm{KT}}(\pi^T, B^T)$ where $\mathrm{d}_{\mathrm{KT}}(\pi^0, B^0) = 0$. □

3.5 Concluding Remarks

In this work we examine the polynomial-time approximability of Multistage Min-Sum Set Cover. We present $\Omega(\log n)$ and $\Omega(r)$ inapproximability results for general and r-bounded request sequences, while we respectively provide $O(\log^2 n)$ and $O(r^2)$ polynomial-time approximation algorithms. Closing this gap is an interesting question that our work leaves open. Another interesting research direction concerns the competitive ratio in the online version of Dynamic Min-Sum Set Cover. [29] provides an $\Omega(r)$ lower bound and a $\Theta\left(r^{3/2}\sqrt{n}\right)$-competitive online algorithm for r-bounded sequences. Designing online algorithms for a relaxation of the problem (such as the Fractional − MTF) and using the rounding schemes that this work suggests may be a fruitful approach towards closing this gap.

Bibliography

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

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

[3] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms (TALG)*, 2(2):153–177, 2006.

[4] Christoph Ambühl. Offline list update is NP-hard. In *Algorithms - ESA 2000, 8th Annual European Symposium, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*, pages 42–51. Springer, 2000.

[5] Hyung-Chan An, Ashkan Norouzi-Fard, and Ola Svensson. Dynamic facility location via exponential clocks. *ACM Trans. Algorithms*, 13(2):21:1–21:20, 2017.

[6] Yossi Azar and Iftah Gamzu. Ranking with submodular valuations. In *SODA*, pages 1070–1079, 2011.

[7] Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. Multiple intents re-ranking. In *STOC*, pages 669–678, 2009.

[8] Nikhil Bansal, Jatin Batra, Majid Farhadi, and Prasad Tetali. Improved approximations for min sum vertex cover and generalized min sum set cover. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 998–1005. SIAM, 2021.

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

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

[11] Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.

[12] Omer Ben-Porat and Moshe Tennenholtz. A game-theoretic approach to recommendation systems with strategic content providers. In *Annual Conference on Neural Information Processing Systems 2018*, NeurIPS 2018, 2018.

[13] Nadja Betzler, Arkadii Slinko, and Johannes Uhlmann. On the computation of fully proportional representation. *Journal of Artificial Intelligence Research*, 47:475–519, 2013.

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

[15] Guillaume Cabanac and Thomas Preuss. Capitalizing on order effects in the bids of peer-reviewed conferences to secure reviews by expert referees. *J. Am. Soc. Inf. Sci. Technol.*, 64(2):405–415, February 2013. `doi:10.1002/asi.22747`.

[16] Ioannis Caragiannis, Laurent Gourvès, and Jérôme Monnot. Achieving proportional representation in conference programs. In *Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, pages 144–150, 2016.

[17] Bart 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.

[18] Mahsa Derakhshan, Negin Golrezaei, Vahideh Manshadi, and Vahab Mirrokni. Product ranking on online platforms. In *Proc. of the 21st ACM Conference on Economics and Computation (EC 2015)*. ACM, 2020. URL: `https://ssrn.com/abstract=3130378`.

[19] Zvi Drezner and Horst W. Hamacher. *Facility location - Applications and Theory*. Springer, 2002. URL: `http://www.springer.com/computer/swe/book/978-3-540-42172-6`.

[20] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 613–622, New York, NY, USA, 2001. Association for Computing Machinery.

[21] David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.

[22] David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In *International Colloquium on Automata, Languages, and Programming*, pages 459–470. Springer, 2014.

[23] Uriel Feige, László Lovász, and Prasad Tetali. Approximating min-sum set cover. In *Approximation Algorithms for Combinatorial Optimization, 5th International Workshop, APPROX 2002, Rome, Italy, September 17-21, 2002, Proceedings*, pages 94–107, 2002. `doi:10.1007/3-540-45753-4\_10`.

[24] Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[25] Cristina Fernandes, Marcio Oshiro, and Nicolas Schabanel. Dynamic clustering of evolving networks: some results on the line. 2013.

[26] Tanner Fiez, Nihar Shah, and Lillian Ratliff. A super* algorithm to determine orderings of items to show users. In *Conference on Uncertainty in Artificial Intelligence*, 2020.

[27] Dimitris Fotakis, Loukas Kavouras, Panagiotis Kostopanagiotis, Philip Lazos, Stratis Skoulakis, and Nikos Zarifis. Reallocating multiple facilities on the line. *Theoretical Computer Science*, 858:13–34, 2021. URL: `https://www.sciencedirect.com/science/article/pii/S0304397521000517`, `doi:https://doi.org/10.1016/j.tcs.2021.01.028`.

[28] Dimitris Fotakis, Loukas Kavouras, Grigorios Koumoutsos, Stratis Skoulakis, and Manolis Vardas. The online min-sum set cover problem. In *Proc. of the 47th International Colloquium on Automata, Languages and Programming (ICALP 2020)*, LIPIcs, 2020.

[29] Dimitris Fotakis, Thanasis Lianeas, Georgios Piliouras, and Stratis Skoulakis. Efficient online learning of optimal rankings: Dimensionality reduction via gradient descent. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

[30] Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 563–575. Springer, 2014.

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

[32] Sungjin Im. Min-sum set cover and its generalizations. In *Encyclopedia of Algorithms*, pages 1331–1334. Springer, 2016.

[33] Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. Minimum latency submodular cover. *ACM Trans. Algorithms*, 13(1):13:1–13:28, 2016.

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

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

[36] Nevena Lazic. *Message Passing Algorithms for Facility Location Problems.* PhD thesis, University of Toronto, 2011.

[37] Alejandro López-Ortiz, Marc P. Renault, and Adi Rosén. Paid exchanges are worth the price. *Theoretical Computer Science*, 824-825:1–10, 2020.

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

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

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

[41] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[42] Matthew J. Streeter, Daniel Golovin, and Andreas Krause. Online learning of assignments. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009*, pages 1794–1802. Curran Associates, Inc., 2009.

[43] Erez Timnat. The list update problem, 2016. Master Thesis, Technion- Israel Institute of Technology.