Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Άμεσοι Αλγόριθμοι Επαυξημένοι με Προβλέψεις για Προβλήματα Χωροθέτησης και Ανάθεσης Πόρων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΩΝ ΧΑΤΖΗΘΕΟΔΩΡΟΥ

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

Αθήνα, Μάρτιος 2023

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Άμεσοι Αλγόριθμοι Επαυξημένοι με Προβλέψεις για Προβλήματα Χωροθέτησης και Ανάθεσης Πόρων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΩΝ ΧΑΤΖΗΘΕΟΔΩΡΟΥ

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Μαρτίου 2023.

.....	.....	.....
Δημήτριος Φωτάκης Αν. Καθηγητής Ε.Μ.Π.	Αριστείδης Παγουρτζής Καθηγητής Ε.Μ.Π.	Αντώνης Συμβώνης Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2023

Ιάσων Χατζηθεοδώρου

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

Copyright © Ιάσων Χατζηθεοδώρου, 2023.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Στην παρούσα διπλωματική, εστιάζουμε σε μεθόδους βελτίωσης των αλγορίθμων για άμεσα προβλήματα σε γράφους, χρησιμοποιώντας προβλέψεις. Οι αλγόριθμοι επαυξημένοι με προβλέψεις είναι μια πρόσφατη κατεύθυνση στην προσπάθεια μας να ξεπεράσουμε την ανάλυση χειρότερης περίπτωσης, που δίνει απαισιόδοξα αποτελέσματα για πολλά προβλήματα. Εξαιτίας της πληθώρας των συλλεγμένων δεδομένων, η λήψη προβλέψεων θεωρείται τετριμμένη και μας δίνει την ευκαιρία να πετύχουμε πολύ καλύτερους λόγους προσέγγισης. Οι προβλέψεις ομως που λαμβάνουμε, δίνονται χωρίς εγγυήσεις για την ποιότητά τους, οπότε πρέπει να πετύχουμε μία κατάλληλη ισορροπία ανάμεσα στο να τις εμπιστευόμαστε και να τις αγνοούμε. Συγκεκριμένα, χρησιμοποιούμε έναν αλγόριθμο που σχεδιάστηκε για άμεσα προβλήματα σε γράφους ώστε να επωφεληθούμε από τις προβλέψεις στο πρόβλημα Ενοικίασης Καταστημάτων, μια έκδοση του Προβλήματος Χωροθέτησης, όπου η κατανομή των πελατών μεταβάλλεται με το χρόνο. Στο πρόβλημα αυτό, αντί να αγοράσουμε καταστήματα για να καλύψουμε τους πελάτες, τα νοικιάζουμε για συγκεκριμένα χρονικά διαστήματα. Η χρονική εξάρτηση αποδεικνύεται σημαντική πρόκληση κατά την ανάλυση του αλγορίθμου, καθώς μία πιθανή λύση για τις προβλέψεις δεν μπορεί να προσαρμοστεί με προφανή τρόπο για να καλύψει την είσοδο.

Λέξεις κλειδιά

Αλγόριθμοι επαυξημένοι με προβλέψεις, Πρόβλημα Χωροθέτησης, Πρόβλημα Ενοικίασης Καταστημάτων, Άμεσα προβλήματα γράφων

Abstract

In this thesis, we focus on ways to improve the algorithms for online graph problems, by taking advantage of predictions. Learning-augmented algorithms are a recent direction in the field of Beyond-Worst-Case analysis of algorithms, which aims to overcome known pessimistic bounds for many problems. Due to the abundance of collected data, acquiring predictions is considered standard and gives us the opportunity to achieve much better approximations. Yet, these predictions are given without any guarrantees for their quality, therefore we need to be careful that we do not trust them blindly, while also not ignoring them completely. We proceed to use a framework for online graph problems in order to take advantage of predictions for Online Facility Leasing, a variant of Online Facility Location, where the distribution of the demands also changes with time. In this problem, instead of buying facilities to cover demands, we lease them choosing from k different lease durations. Time dependency proves to be a challenge for the analysis of the algorithm, as a solution for a predicted instance may not be readily adapted for use with the input and requires a problem specific approach.

Key words

Beyond-Worst-Case analysis, Learning-augmented algorithms, Algorithms with predictions, Facility Location, Facility Leasing, Online graph problems

Ευχαριστίες

Πρώτα απ'όλα θα ήθελα να ευχαριστήσω τον κύριο Φωτάκη, όχι μόνο για την υποστήριξή του κατά τη διπλωματική, αλλά και σε όλη τη διάρκεια της σχολής, από τα προπτυχιακά μαθήματά του που μου καλλιέργησαν την περιέργεια για τη Θεωρητική Πληροφορική, μέχρι και τις αιτήσεις μου για διδακτορικές σπουδές στο εξωτερικό. Επίσης, χαίρομαι ιδιαίτερα που είχα την ευκαιρία να ενταχθώ στο study group του εργαστηρίου, όπου πέρα από ενδιαφέροντα θέματα στην Πληροφορική, είχαμε τη δυνατότητα να συζητάμε τις δυσκολίες αλλά και τις επιτυχίες μας στο τελευταίο κομμάτι των σπουδών μας. Σε όλες αυτές τις προσπάθειες βρέθηκαν δίπλα μου η οικογένειά μου, παρέχοντας συνεχή στήριξη και οι φίλοι μου, οι οποίοι έκαναν τα χρόνια των προπτυχιακών σπουδών μου πολύ πιο ευχάριστα από ό,τι θα μπορούσα να φανταστώ.

Ιάσων Χατζηθεοδώρου,

Αθήνα, 30η Μαρτίου 2023

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εκτεταμένη Ελληνική Περίληψη

1.1 Εισαγωγή

Η πιο θεμελιώδης πρόκληση που αντιμετωπίζουμε στη μελέτη των άμεσων αλγορίθμων είναι η αβεβαιότητά μας για τα μελλοντικά δεδομένα, στα οποία δεν έχουμε ακόμα πρόσβαση. Αυτό γίνεται κατανοητό αν σκεφτούμε την περίπτωση στην οποία κάποιος αντίπαλος μας παρέχει τα δεδομένα με σκοπό να μας αναγκάσει να κάνουμε λάθος επιλογές. Η παραπάνω μέθοδος ανάλυσης αλγορίθμων λέγεται Worst-Case και αποτελεί την κλασική προσέγγιση για τον χαρακτηρισμό των αλγορίθμων και κατ'επέκταση των προβλημάτων. Δυστυχώς, με αυτόν τον τρόπο οι λόγοι προσέγγισης που παίρνουμε είναι αρκετά απαισιόδοξοι. Μερικά παραδείγματα βέλτιστων αλγορίθμων που αναδεικνύουν αυτό το πρόβλημα είναι ο ντετερμινιστικός $O(k)$ και πιθανοτικός $O(\log k)$ προσεγγιστικοί αλγόριθμοι για Online Paging [Fiat et al., 1991], ο $O(\log n)$-προσεγγιστικός αλγόριθμος για Online Steiner Tree [Imase and Waxman, 1991] και ο $\frac{\log n}{\log \log n}$ προσεγγιστικός αλγόριθμος για Online Facility Location [Fotakis, 2011].

Μία νέα προσέγγιση που προτάθηκε από τους [Lykouris and Vassilvitskii, 2018] για την αντιμετώπιση των δυσκολιών που επιφέρει η αβεβαιότητα, είναι η πρόβλεψη μελλοντικών χαρακτηριστικών των δεδομένων εισόδου και η χρήση τους κατά την εκτέλεση. Με αυτόν τον τρόπο ξεκίνησε η μελέτη των αλγορίθμων επαυξημένων με προβλέψεις και μέσα σε λίγα χρόνια έχει οδηγήσει σε πλήθος από αποτελέσματα. Υπό μία έννοια, η επιτυχία των μοντέλων μηχανικής μάθησης παρέχουν σημαντικό κίνητρο σε αυτή την προσπάθεια, καθώς εξασφαλίζουν την απόκτηση των προβλέψεων. Φυσικά, τα παραπάνω μοντέλα δεν περιμένουμε να δίνουν τέλειες προβλέψεις, παρά μόνο αρκετά καλές ώστε να πετυχαίνουμε βελτιώσεις. Οπότε είναι λογικό να κατέχει κεντρική θέση η έννοια του σφάλματος των προβλέψεων κατά την ανάλυση. Διαισθητικά, θα θέλαμε να πετυχαίνουμε λύσεις πολύ κοντά στις βέλτιστες όταν οι προβλέψεις έχουν χαμηλό σφάλμα, ενώ στην αντίθετη περίπτωση θα θέλαμε να μην είναι χειρότερες από αυτές των ήδη γνωστών αλγορίθμων που δεν έχουν πρόσβαση σε προβλέψεις. Φυσικά, αυτό δεν θα είναι γνωστό κατά την εκτέλεση, καθώς εξαρτάται από ολόκληρη την άγνωστη είσοδο. Ένα ακόμα πιο κρίσιμο ερώτημα είναι το τι θα επιλέξουμε ως κατάλληλη πρόβλεψη. Σε κάποιες περιπτώσεις αρκεί να περιοριστούμε στα δεδομένα που θα χρησιμοποιούσε ένας βέλτιστος αλγόριθμος που γνωρίζει όλη την είσοδο, όπως κάνουν οι [Lykouris and Vassilvitskii, 2018] για το Online Paging. Μία απλούστερη επιλογή είναι να προβλέψουμε ολόκληρη την είσοδο εξ αρχής, όπως για παράδειγμα κάνουν οι [Xu and Moseley, 2021, Azar et al., 2021]. Επιπλέον έχουν χρησιμοποιηθεί προβλέψεις σχετικές με τη μορφή της βέλτιστης λύσης [Bamas et al., 2020, Fotakis et al., 2021, Jiang et al., 2020, Agrawal et al., 2022].

Το πεδίο των άμεσων αλγορίθμων μέχρι τώρα έχει συναντήσει τη μεγαλύτερη επιτυχία στη χρήση προβλέψεων, με μεγάλο πλήθος προβλημάτων να έχουν μελετηθεί από αυτή την οπτική γωνία. Συγκεκριμένα, έχει μελετηθεί το Online Paging [Lykouris and Vassilvitskii, 2018, Rohatgi, 2019, Antoniadis et al., 2020] και η γενίκευσή του Weighted Online Paging [Jiang et al., 2020, Bansal et al., 2020]. Επίσης, έχει μελετηθεί το Non-Clairvoyant Scheduling σε διάφορες εκδοχές [Purohit et al., 2018, Mitzenmacher, 2019, Im et al., 2021a, Cho et al., 2022, Lindermayr and Megow, 2022, Zhao et al., 2022]. Μία άλλη κατηγορία που έχει μελετηθεί εκτενώς είναι τα

προβλήματα δικτύων, όπως το Online Steiner Tree [Xu and Moseley, 2021] και Online Facility Location [Almanza et al., 2021, Fotakis et al., 2021, Jiang et al., 2021]. Επιπλέον, υπάρχουν εφαρμογές σε κλασικά προβλήματα όπως το Online Subset Sum [Xu and Zhang, 2022], Online Knapsack [Thang and Durr, 2021, Im et al., 2021b, Boyar et al., 2022].

Σε συνδυασμό με τις παραπάνω λύσεις σε συγκεκριμένα προβλήμα, έχουν αναπτυχθεί και γενικότερες μέθοδοι, με τις οποίες μπορούμε να αντιμετωπίσουμε ολόκληρες κατηγορίες προβλημάτων. Συγκεκριμένα, η μέθοδος primal-dual γενικεύτηκε ώστε να συμπεριλάβει την χρήση προβλέψεων για τη λύση [Bamas et al., 2020] και εφαρμόστηκε στα προβλήματα Online Weighted Set Cover, Ski Rental, TCP Acknowledgment. Εξετάστηκε επίσης και η περίπτωση που μας δίνονται πολλαπλές προβλέψεις για την κατηγορία των covering problems, πχ Online Facility Location κατά τη διάρκεια της εκτέλεσης από τους [Anand et al., 2022a]. Μια ακόμα προσέγγιση είναι αυτή των [Azar et al., 2021] που σχεδιάστηκε για προβλήματα σε γράφους, όπως το Online Steiner Tree, δεδομένων προβλέψεων ολόκληρης της εισόδου, η επιτυχία της οποίας συνίσταται στον ορισμό μίας γενικευμένης έννοιας σφάλματος. Όπως είναι φανερό, αυτές οι μέθοδοι καλύπτουν πολλές κατηγορίες προβλημάτων αλλά και διαφορετικά είδη προβλέψεων.

Βασιζόμενοι στην τελευταία από αυτές [Azar et al., 2021], θα προσπαθήσουμε να χρησιμοποιήσουμε προβλέψεις για την επίλυση του προβλήματος Online Facility Leasing [Anthony and Gupta, 2007], το οποίο συνδυάζει τη χωρική κατανομή του Facility Location [Meyerson, 2001] με τη χρονική κατανομή του Parking Permit [Meyerson, 2005]. Όπως θα φανεί στο τελευταίο κεφάλαιο, στην περίπτωση που μας δίνονται καλές προβλέψεις, ο αλγόριθμος ξεπερνάει το βέλτιστο κόστος μόνο κατά μία σταθερά, ενώ στην περίπτωση που οι προβλέψεις αποδειχθούν αναξιόπιστες πετυχαίνει την ίδια προσέγγιση με τον αλγόριθμο που δεν χρησιμοποιεί προβλέψεις [Nagarajan and Williamson, 2013].

1.2 Ορισμοί

Για την ανάλυση των αλγορίθμων που θα παρουσιαστούν είναι απαραίτητες κάποιες βασικές έννοιες. Συγκεκριμένα, οι [Lykouris and Vassilvitskii, 2018] ανέλυσαν τους αλγορίθμους τους επεκτείνοντας την έννοια του λόγου προσέγγισης, ώστε να εκφράσει την εξάρτηση από το σφάλμα των προβλέψεων.

Ορισμός 1.2.1 (Λόγος Προσέγγισης)**.** *Ένας αλγόριθμος A έχει λόγο προσέγγισης c αν για κάθε στιγμιότυπο σ ισχύει*

$$cost_A(\sigma) \leq c \cdot OPT(\sigma)$$

Οι δύο εγγυήσεις που θα πρέπει να δίνει ένας αλγόριθμος που χρησιμοποιεί προβλέψεις σχετίζονται με την ποιότητα της λύσης, όταν οι προβλέψεις είναι σωστές και όταν οι προβλέψεις είναι εντελώς λανθασμένες.

Ορισμός 1.2.2 (Συνέπεια)**.** *Ένας αλγόριθμος είναι β-συνεπής εάν ο λόγος προσέγγισής του είναι β όταν το σφάλμα των προβλέψεω είναι το ελάχιστο δυνατό, δηλαδή οι προβλέψεις είναι απόλυτα σωστές.*

Ορισμός 1.2.3 (Ευρωστία)**.** *Ένας αλγόριθμος είναι α-συνεπής εάν ο λόγος προσέγγισής του είναι α όταν το σφάλμα των προβλέψεω είναι το μέγιστο δυνατό, δηλαδή ανεξάρτητα με την ποιότητα των προβλέψεων.*

Χρησιμοποιώντας τους παραπάνω ορισμούς, τα επιθυμητά χαρακτηριστικά ενός αλγορίθμου με προβλέψεις θα είναι η σταθερή συνέπεια και ευρωστία που δεν ξεπερνά κατά πολύ τον καλύτερο αλγόριθμο χωρίς προβλέψεις.

Όσον αφορά στους αλγόριθμους που θα χρησιμοποιηθούν, μια χρήσιμη ιδιότητα για τον συνδυασμό τους με άλλους είναι να διατηρούν το λόγο προσέγγισής τους σε κάθε υποσύνολο των δεδομένων εισόδου [Azar et al., 2021].

Ορισμός 1.2.4 (Subset Competitive)**.** *Ένας άμεσος αλγόριθμος ON με είσοδο R και λόγο προσέγγισης $f(|R|)$ είναι subset-competitive εάν για κάθε $R' \subseteq R$ το κόστος που πληρώνει στα βήματα που του έρχονται τα στοιχεία R' είναι*

$$ON(R') \leq f(|R'|) \cdot OPT(R')$$

Τέλος, είναι σκόπιμο να αναφερθούμε στις prize-collecting εκδοχές των προβλημάτων. Αποτελούν παραλλαγές των άμεσων προβλημάτων στις οποίες δεν χρειάζεται να καλύψουμε κάθε στοιχείο της εισόδου με κάποια λύση, αλλά έχουμε την επιλογή να πληρώσουμε ένα γνωστό από πριν κόστος και να το αφήσουμε ακάλυπτο. Διαισθητικά, με αυτόν τον τρόπο έχουμε τη δυνατότητα να αποκτήσουμε λύσεις επιθυμητού κόστους, οι οποίες καλύπτουν τα μέρη της εισόδου που θέλουμε.

Ορισμός 1.2.5 (Prize-collecting)**.** *Αν υποθέσουμε ότι ένας άμεσος αλγόριθμος για ένα πρόβλημα με είσοδο R πληρώνει $c(r)$ για να καλύψει ένα $r \in R$ αγοράζοντας μια λύση για αυτό, ή να τιμωρηθεί με κόστος $\pi(r)$, όπου η π είναι γνωστή εξ αρχής. Τότε στόχος είναι να ελαχιστοποιήσουμε το*

$$\sum_{r \in S} c(r) + \sum_{r \in R \setminus S} \pi(r)$$

όπου $S \subseteq R$ είναι το σύνολο των καλυμμένων $r \in R$.

1.3 Γενικές Μέθοδοι

Παρόλο που για μεγάλο αριθμό προβλημάτων η χρήση των προβλέψεων γίνεται με τρόπο προσαρμοσμένο στο συγκεκριμένο πρόβλημα, πρόσφατα έχουν αναπτυχθεία και γενικότερες μέθοδοι, οι οποίες λειτουργούν με τη λογική του μαύρου κουτιού και καλύπτουν ευρύτερες κατηγορίες προβλημάτων. Σε αυτή την ενότητα θα παρουσιάσουμε σύντομα τρία από αυτά, με κριτήριο το εύρος των προβλημάτων στα οποία μπορούν να εφαρμοστούν.

1.3.1 Primal Dual Επαυξημένο με Προβλέψεις

Η τεχνική primal dual αναπτύχθηκε για την επίλυση του κλασματικού Weighted Set Cover [Alon et al., 2003] βρήκε εφαρμογή σε μεγάλο πλήθος προβλημάτων. Η βασική ιδέα είναι πως διατηρούμε μια εφικτή λύση για το πρωτεύον πρόβλημα και μία μη εφικτή για το δυϊκό, που απέχει ένα συγκεκριμένο παράγοντα από το να γίνει εφικτή. Εάν η λύση του πρωτεύοντος φράσσεται άνω από τη λύση του δυϊκού επί έναν άλλο παράγοντα τότε από ασθενή δυϊκότητα αποκτούμε το λόγο προσέγγισης. Το επόμενο βήμα είναι να μετατρέψουμε αυτή τη λύση σε ακέραια χρησιμοποιώντας κάποια τεχνική στρογγυλοποίησης που επιφέρει τον παράγοντα του integrality gap.

Η ίδια ακριβώς λογική ακολουθείται και στην περίπτωση του primal dual με προβλέψεις, PDLA [Bamas et al., 2020]. Η μόνη διαφορά είναι ότι κατά την ενημέρωση των μεταβλητών του πρωτεύοντος θέλουμε να αυξάνουμε πιο έντονα αυτές που υποστηρίζονται από τις προβλέψεις, οι οποίες αναφέρονται στην βέλτιστη λύση.

Weighted Set Cover με Προβλέψεις. Το πρόβλημα που χρησιμοποιείται ως παράδειγμα για τη συγκεκριμένη μέθοδο είναι το Online Weighted Set Cover. Δεδομένου ενός συνόλου $\mathcal{U} = \{e_1, e_2, \ldots, e_n\}$ και ένα σύνολο $\mathcal{F}$ από υποσύνολα με βάρη του $\mathcal{U}$. Σκοπός μας είναι να καλύψουμε κάθε στοιχείο του $\mathcal{U}$ αγοράζοντας x_S ποσοστό από κάθε $S \in \mathcal{F}$ δηλαδή $\sum_{S \in \mathcal{F}(e)} x_S \geq 1$, ενώ ελαχιστοποιούμε το ολικό κόστος $\sum_{S \in \mathcal{F}} w_S \cdot x_S$. Οι προβλέψεις που μας δίνονται είναι ένα υποσύνολο $\mathcal{A} \subseteq \mathcal{F}$ που αντιστοιχεί στην βέλτιστη ακέραιη λύση.

Algorithm 1: PDLA Weighted Set Cover

forall *e in the input* **do**
  while $\sum_{S \in F(e)} x_S < 1$ **do**
    forall $S \in F(e)$ **do**
      $x_S \leftarrow x_S \cdot (1 + \frac{1}{w_S}) + \frac{\lambda}{w_S |\mathcal{F}(e)|} + \frac{1-\lambda}{w_S |\mathcal{F}(e) \cap \mathcal{A}|} \cdot 1\{S \in \mathcal{A}\}$
    $y_e \leftarrow y_e + 1$

Θεώρημα 1.3.1. *Το κόστος του αλγορίθμου είναι* $ALG \leq \min\{\Theta(\log(d/\lambda)) \cdot OPT, O(\frac{1}{1-\lambda}) \cdot S(\mathcal{A}, \mathcal{I})\}$*, όπου* $S(\mathcal{A}, \mathcal{I})$ *είναι το κόστος της λύσης που δίνουν οι προβλέψεις.*

1.3.2 Προβλήματα Κάλυψης με Πολλαπλές Προβλέψεις

Παραμένοντας στην ίδια περιοχή προβλημάτων, εστιάζουμε σε μια γενικότερη περίπτωση, τα άμεσα προβλήματα κάλυψης. Αυτά τα προβλήματα περιέχουν το Online Set Cover που είδαμε προηγουμένως και με μικρές τροποποιήσεις ακόμα και το Online Facility Location. Όπως έδειξαν οι [Anand et al., 2022a], μπορούμε να εκμεταλλευτούμε πολλαπλές προβλέψεις σε κάθε βήμα και να συγκρίνουμε την λύση μας με τον καλύτερο συνδυασμό τους.

Άμεσο Πρόβλημα Κάλυψης. Έστω ότι μας δίνονται κόστη $c_i \geq 0$ και συντελεστές $a_{ij} \geq 0$. Τότε το γραμμικό πρόγραμμα που περιγράφει το πρόβλημα έχει την εξής μορφή

$$\begin{aligned} \min & \sum_i c_i \cdot x_i \\ s.t. & \sum_i a_{ij} \cdot x_i \geq 1 \quad \forall j \in [m] \\ & x_i \in [0,1] \quad \forall i \in [n] \end{aligned}$$

Όπου οι c_i δίνονται στην αρχή και σε κάθε βήμα j αποκαλύπτεται ένας περιορισμός $\sum_i a_{ij} \cdot x_i \geq 1$, ο οποίος πρέπει να ικανοποιηθεί στο ίδιο βήμα και για όλα τα επόμενα. Αυτό μπορούμε να το πετύχουμε αποφασίζοντας να μην μειώνουμε τις τιμές των μεταβλητών.

Προβλέψεις. Σε κάθε βήμα j δίνονται k προβλέψεις για τις τιμές των μεταβλητών. Η s πρόβλεψη για την i-οστή μεταβλητή θα είναι η $\hat{x}_i(j, s)$ και το μόνο που γνωρίζουμε για αυτές είναι πως ικανοποιούν τον περιορισμό αυτού του βήματος.

Για να φράξουμε το κόστος του αλγορίθμου θα χρησιμοποιηθούν δύο λύσεις που ορίζονται αποκλειστικά για τις προβλέψεις. Συγκεκριμένα, η πρώτη απλά θα υπολογίζει το κόστος που θα είχαμε αν ακολουθούσαμε τυφλά την καλύτερη πρόβλεψη, δηλαδή

$$STATIC = \min_{s \in [k]} \sum_{i \in [n]} c_i \cdot \max_{j \in [m]} \hat{x}_i(j, s)$$

Από την άλλη, μία καλύτερη λύση είναι αυτή που παίρνουμε αν θεωρήσουμε ξεχωριστά τις προβλέψεις που δίνονται σε κάθε βήμα και βρούμε την καλύτερη από αυτές. Το σύνολο των διαθέσιμων επιλογών τότε θα είναι $\hat{X} = \{\hat{x} \in [0,1]^n \mid \forall i \in [n], \forall j \in [m], \exists s \in [k] : \hat{x}_i \geq \hat{x}_i(j, s)\}$ και η λύση θα είναι

$$DYNAMIC = \min_{\hat{x} \in \hat{X}} \sum_{i \in [n]} c_i \cdot \hat{x}_i$$

Όπως είναι φανερό, οι διαθέσιμες επιλογές της $DYNAMIC$ περιέχουν τις επιλογές της $STATIC$ οπότε $DYNAMIC \leq STATIC$.

Άμεσος Αλγόριθμος για το Πρόβλημα Κάλυψης με Προβλέψεις. Σε κάθε βήμα θα αυξάνουμε τις μεταβλητές με διαφορετικούς ρυθμούς. Όπως είναι λογικό, αυτή η αύξηση θα είναι ανάλογη της συνεισφοράς της μεταβλητής i στην κάλυψη του περιορισμού, a_{ij} και αντιστρόφως ανάλογη στο κόστος c_i που επιφέρει. Για να εισάγουμε την επιρροή των προβλέψεων, θα θεωρήσουμε το μέσο όρο των προβλέψεων που δόθηκαν στο συγκεκριμένο βήμα για την κάθε μεταβλητή, δηλαδή $\frac{1}{k}\sum_s \hat{x}_i(j,s)$. Στον αλγόριθμο συμβολίζουμε $\delta = \frac{1}{k}$ και $x_{ij} = \sum_s \hat{x}_i(j,s)$.

Algorithm 2: Online Covering Framework

```
forall steps j do
    while ∑_{i∈[n]} a_ij x_i < 1/2 do
        forall i ∈ [n] do
            if x_i < 1/2 then
                Increase x_i with rate dx_i/dt = (a_ij / c_i) · (x_i + δ · x_ij)
```

Συνδυάζοντας τον παραπάνω αλγόριθμο με έναν οποιονδήποτε άμεσο χωρίς προβλέψεις παίρνουμε το παρακάτω θεώρημα. Ο συνδυασμός μπορεί να γίνει χρησιμοποιώντας τον ίδιο αλγόριθμο και δίνοντας του ως πρώτη πρόβλεψη τις λύσεις του instance που έχει πρόσβαση σε προβλέψεις και ως δεύτερη τις λύσεις του άμεσου χωρίς προβλέψεις.

Θεώρημα 1.3.2. *Υπάρχει αλγόριθμος του οποίου το κόστος είναι* $ALG \leq \min\{O(\log k) \cdot DYNAMIC, O(\log d)OPT\}$*, όπου* d *είναι το μέγιστο πλήθος μεταβλητών με μη μηδενικούς συντελεστές σε κάθε περιορισμό.*

1.3.3 Άμεσοι Αλγόριθμοι σε Γράφους με Προβλέψεις

Στην κατηγορία των προβλημάτων που ορίζονται σε μετρικούς χώρους (ή σε γράφους με την συνήθη έννοια απόστασης), ένα κρίσιμο χαρακτηριστικό είναι η χωρική κατανομή των δεδομένων, τα οποία εμφανίζονται ένα ένα και πρέπει να καλυφθούν. Οπότε στην περίπτωση που η πρόβλεψη είναι ολόκληρη η είσοδος, η ποιότητά της θα καθοριστεί από τη σύγκριση της κατανομής της με την πραγματική είσοδο. Το πιο γενικό πλαίσιο μέχρι τώρα που καλύπτει αυτή την περίπτωση είναι το [Azar et al., 2021], καθώς σχεδιάστηκε με τη λογική να απορρίπτει εντελώς λανθασμένες προβλέψεις αλλά και να συγκρίνει τις κατανομές των κοντινών προβλέψεων με την είσοδο ως προς την απόστασή τους. Παραδείγματα προβλημάτων στα οποία μπορεί να χρησιμοποιηθεί είναι το Online Steiner Tree/Forest και Online Facility Location.

Προβλέψεις και Σφάλματα. Όπως ήδη αναφέρθηκε, η πρόβλεψη $\hat{R}$ αντιστοιχεί σε ολόκληρη την είσοδο R και δίνεται εξ αρχής. Για τον υπολογισμό του σφάλματος χρησιμοποιείται μια δυάδα (Δ, D), όπου το Δ περιγράφει το πλήθος των προβλέψεων/δεδομένων που είναι πολύ μακριά για να αντιστοιχιστούν και D είναι το κόστος για την αντιστοίχιση των υπολοίπων. Πιο συγκεκριμένα, εάν $T \subseteq R$ και $\hat{T} \subseteq \hat{R}$ είναι τα υποσύνολα των δύο κατανομών που βρίσκονται αρκετά κοντά ώστε να έχει νόημα η αντιστοίχισή τους, τότε $\Delta = |R \setminus T \uplus \hat{R} \setminus \hat{T}|$ και το D υπολογίζεται ανάλογα με το πρόβλημα, τυπικά με min cost matching ανάμεσα στα $T, \hat{T}$. Όπως φαίνεται, υπάρχουν πολλοί πιθανοί συνδυασμοί αυτών των δύο και τα αποτελέσματα ισχύουν για οποιονδήποτε από αυτούς, αν και τυπικά τα καλύτερα bounds προκύπτουν στο Pareto frontier.

Απαιτήσεις. Ο συγκεκριμένος αλγόριθμος λειτουργεί συνδυάζοντας δύο άλλους. Ο πρώτος από αυτούς θα είναι ένας οποιοσδήποτε άμεσος αλγόριθμος για το πρόβλημα, με τον περιορισμό ότι πρέπει να είναι subset competitive και θα χρησιμοποιηθεί για να καλύπτει τα δεδομένα της εισόδου. Ο δεύτερος θα είναι offline και θα χρησιμοποιείται για να αγοράζει μερικές λύσεις πάνω στις προβλέψεις, για αυτό και χρειάζεται να είναι prize-collecting και σταθερού λόγου προσέγγισης. Σκοπός του συνολικού αλγορίθμου είναι να δουλεύει σαν να γνώριζε τα υποσύνολα $T, \hat{T}$. Εάν ίσχυε αυτό τότε θα μπορούσε να τρέξει έναν offline αλγόριθμο πάνω στο $\hat{T}$ και πληρώνοντας ένα κόστος αντιστοίχισης να χρησιμοποιήσει αυτή τη λύση για να καλύψει το T.

Για την υπόλοιπη είσοδο θα αρκούσε να χρησιμοποιήσει έναν online αλγόριθμο. Το εντυπωσιακό είναι πως ενώ δεν μπορεί να ακολουθήσει αυτή την τακτική, καθώς τα $T, \hat{T}$ δεν είναι γνωστά, πετυχαίνει τελικά τα ίδια αποτελέσματα.

Αλγόριθμος. Η βασική ιδέα που χρησιμοποιείται είναι πως μπορούμε να προσθέτουμε λύσεις που καλύπτουν ικανοποιητικά πολλές προβλέψεις χωρίς να ξεπερνάνε ένα όριο κόστους. Σε αυτό ακριβώς βοηθούν οι prize-collecting εκδοχές των προβλημάτων, καθώς έτσι μπορούμε ρυθμίζοντας τα πρόστιμα, να βρίσκουμε τα μέρη των προβλέψεων που μας συμφέρει να καλύψουμε. Από εκεί και πέρα αρκεί να προχωρούμε με εκθετικά βήματα το όριο κόστους $\hat{B}_i$ του άμεσου αλγορίθμου και κάθε φορά που το φτάνει να προσθέτουμε λύσεις ίδιου κόστους που καλύπτουν πολλές προβλέψεις. Ο πλήρης αλγόριθμος παρουσιάζεται στο αγγλικό κείμενο.

Τα παρακάτω δύο λήμματα χωρίζουν την ανάλυση σε δύο μέρη με βάση την στιγμή i η οποία επιλέγεται έτσι ώστε να έχει καλυφθεί $|T|$ μέρος της εισόδου.

Λήμμα 1.3.1. *Το κόστος των επαναλήψεων μέχρι την i είναι $Prefix(i) \leq O(1) \cdot OPT + (12\gamma + 2) \cdot \hat{B}_{i-1}$*

Λήμμα 1.3.2. *Το κόστος των επαναλήψεων ξεκινώντας με την $i+1$ είναι $Suffix(i) \leq O(1) \cdot \max\{ON_{m-1}, ON_m\}$*

Όπως φαίνεται για την εφαρμογή του αλγορίθμου σε οποιοδήποτε πρόβλημα αρκεί να φράξουμε το $\hat{B}_{i-1}$ καθώς και το κόστος του άμεσου αλγορίθμου στην τελευταία φάση m.

1.4 Αλγόριθμοι Επαυξημένοι με Προβλέψεις

Στην παρούσα ενότητα θα παρουσιάσουμε σύντομα κάποια από τα αποτελέσματα που έχουν επιτευχθεί με τη χρήση προβλέψεων σε προβλήματα που κατέχουν κεντρική σημασία στο χώρο των άμεσων αλγορίθμων.

1.4.1 Ski Rental

Το πρόβλημα Ski Rental είναι ένα από τα απλούστερα προβλήματα στην κατηγορία των άμεσων αλγορίθμων, αλλά την ίδια στιγμή μας δίνει μια καλή εικόνα για τις τεχνικές που χρησιμοποιούνται κατά τη χρήση προβλέψεων. Σκοπός του προβλήματος είναι να επιλέξουμε ανάμεσα στο να νοικιάσουμε με κόστος 1 ανά μέρα ή να αγοράσουμε ένα σετ για σκι με κόστος b. Το ενδιαφέρον είναι πως δεν γνωρίζουμε πόσες μέρες x θα μείνουμε, το οποίο αποτελεί και την πρόβλεψη $\hat{x}$. Ο βέλτιστος άμεσος αλγόριθμος χωρίς προβλέψεις απλά νοικιάζει μέχρι τη μέρα $b-1$ και αγοράζει την επόμενη, οπότε πετυχαίνει 2-προσέγγιση.

Μοναδική Πρόβλεψη. Εφόσον η πρόβλεψη μπορεί να έχει μη φραγμένο σφάλμα, δεν μπορούμε να την εμπιστευθούμε πλήρως ώστε να αγοράζουμε όταν $\hat{x} \geq b$ και να νοικιάζουμε στην αντίθετη περίπτωση. Αυτό μας οδηγεί στο να θέσουμε ένα όριο λb πριν το οποίο δεν αγοράζουμε και b/λ μετά το οποίο σταματάμε να νοικιάζουμε. Το $\lambda \in (0,1)$ μας δίνει τη δυνατότητα να προσαρμόσουμε την εμπιστοσύνη μας στις προβλέψεις.

Θεώρημα 1.4.1. *[Purohit et al., 2018] Ο αλγόριθμος πετυχαίνει $ALG \leq \min\{(1+\frac{1}{\lambda}) \cdot OPT, (1+\lambda) \cdot OPT + \frac{\eta}{1-\lambda}\}$*

Πολλαπλές Προβλέψεις. Ας υποθέσουμε τώρα ότι έχουμε στη διάθεσή μας k προβλέψεις $\hat{x}_i$. Θα ορίσουμε περιοχές απόφασης για τη μέρα που θα αγοράσουμε, έτσι ώστε να ισομοιράζουν το λόγο προσέγγισης. Όσο έχουμε προλέψεις μέσα σε αυτές τις περιοχές, τότε θα νοικιάζουμε και μόλις βρούμε μια κενή περιοχή θα αγοράσουμε και έτσι θα συμπεριλάβουμε και το σφάλμα στην πρόβλεψη. Χρησιμοποιώντας και την παραπάνω τεχνική που δεν επιτρέπουμε αγορά πριν τη μέρα λb έχουμε τα εξής όρια x_i

$$\frac{b+\lambda b}{x_1} = \frac{b+x_1}{x_2} = \cdots = \frac{b+x_{k-1}}{b}$$

Θεώρημα 1.4.2. *[Gollapudi and Panigrahi, 2019] Ο αλγόριθμος πετυχαίνει λόγο προσέγγισης* $\min\{1+\frac{1}{\lambda}, \bar{c}_k\}$ *όπου* $\bar{c}_k = \sum_{i=1}^{k-1} \bar{c}_k^{-i} + \lambda \cdot \bar{c}_k^{-k}$

1.4.2 Non-Clairvoyant Job Scheduling

Σε αυτή την υποενότητα θα παρουσιάσουμε ένα αποτέλεσμα για το πρόβλημα δρομολόγησης εργασιών. Αν υποθέσουμε ότι έχουμε ένα μηχάνημα και n εργασίες που απαιτούν χρόνους εκτέλεσης $x_1, x_2, \ldots, x_n$, θα επιδιώξουμε να ελαχιστοποιήσουμε το άθροισμα των χρόνων ολοκλήρωσης κάθε εργασίας. Όπως είναι φανερό, για να το πετύχουμε αυτό θα θέλαμε οι συντομότερες εργασίες να εκτελεστούν στην αρχή. Όμως στην εκδοχή του προβλήματος που θα μελετήσουμε, οι χρόνοι εκτέλεσης δεν είναι γνωστοί και τους μαθαίνουμε για κάθε εργασία μόνο όταν ολοκληρωθεί η εκτέλεσή της. Ο βέλτιστος αλγόριθμος για αυτό το πρόβλημα είναι ο Round Robin, που απλά εκτελεί τις k εργασίες που απομένουν κάθε στιγμή ενναλάξ, δίνοντας στην κάθε μία $\frac{1}{k}$ ενός ελάχιστου χρονικού παραθύρου πριν προχωρήσει στην επόμενη. Αυτός ο αλγόριθμος είναι 2-προσεγγιστικός.

Αν υποθέσουμε ότι έχουμε στη διάθεσή μας προβλέψεις $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n$, θα τις χρησιμοποιήσουμε έτσι ώστε να προσαρμόσουμε τον ρυθμό $\frac{1}{k}$ με τον οποίο τρέχουμε κάθε εργασία. Στην πραγματικότητα, θα εκτελούμε τις εργασίες βάσει ενός α-προσεγγιστικού αλγορίθμου με αργότερο ρυθμό κατά $1-\lambda$ και βάσει της διάταξης των προβλέψεων με ρυθμό αργότερα κατά λ. Έστω ότι ο δεύτερος αλγόριθμος είναι β-προσεγγιστικός από μόνος του.

Λήμμα 1.4.1. *Ο παραπάνω συνδυασμός πετυχαίνει λόγο προσέγγισης* $\min\{\frac{\beta}{\lambda}, \frac{\alpha}{1-\lambda}\}$.

Ο αλγόριθμος που εμπιστεύεται τις προβλέψεις, SPFJ, πετυχαίνει λόγο προσέγγισης $1+\frac{2\eta}{n}$, όπου το σφάλμα $\eta = \sum_i |x_i - \hat{x}_i|$.

Θεώρημα 1.4.3. *[Purohit et al., 2018] Ο αλγόριθμος που συνδυάζει το Round Robin με το SPJF πετυχαίνει λόγο προσέγγισης* $\min\{\frac{1}{\lambda}(1+\frac{2\eta}{n}), \frac{2}{1-\lambda}\}$.

1.4.3 Άμεσο Δέντρο Steiner

Το Δέντρο Steiner αποτελεί ένα από τα κλασσικά προβλήματα δικτύων. Θεωρώντας ότι έχουμε ένα γράφο G, η είσοδος προσδιορίζει τις τερματικές κορυφές, οι οποίες πρέπει να συνδεθούν. Μία εφικτή λύση, αποτελείται από ακμές οι οποίες δίνουν συνδετικό δέντρο ανάμεσα σε αυτές τις κορυφές και πιθανώς κάποιες από τις υπόλοιπες αν χρειάζονται. Σκοπός είναι να βρούμε το συνδετικό δέντρο ελάχιστου κόστους. Οι προβλέψεις $\hat{R}$ που θα χρησιμοποιηθούν και στις δύο περιπτώσεις θα αφορούν ολόκληρη την είσοδο R και θα δίνονται εξ αρχής.

Μετρώντας το σφάλμα με τις λανθασμένες προβλέψεις Σε αυτή την υποενότητα θα εξετάσουμε ένα αποτέλεσμα, το οποίο αντιμετωπίζει τις προβλέψεις ως απόλυτα σωστές ή λανθασμένες. Όσες προβλέψεις, δηλαδή, δεν ταυτίζονται με κάποια τερματική κορυφή της εισόδου συνεισφέρουν κατά 1 στο σφάλμα. Σε γενικές γραμμές, αν ένα τερματικό r της εισόδου δεν ανήκει στις προβλέψεις, τότε αρκεί να το καλύψουμε όπως ο βέλτιστος άμεσος αλγόριθμος, δηλαδή αγοράζοντας το συντομότερο μονοπάτι που το συνδέει με τα υπόλοιπα. Στην περίπτωση που ανήκει στις προβλέψεις θα θέλαμε να χρησιμοποιήσουμε ένα μονοπάτι του ελάχιστου συνδετικού δέντρου του $\hat{R}$. Όμως αυτό μπορεί να οδηγήσει σε υπερβολικά μεγάλο κόστος, οπότε είναι προτιμότερο να δοκιμάσουμε να βρούμε μονοπάτι σε αυτό το δέντρο με κόστος συγκρίσιμο με αυτό της ακμής που επιλέγαμε αν δεν είχαμε προβλέψεις και αν δεν αρκεί για τη σύνδεση να αγοράσουμε την ακμή.

Θεώρημα 1.4.4. *[Xu and Moseley, 2021] Ο παραπάνω αλγόριθμος πετυχαίνει λόγο προσέγγισης* $O(\log \eta)$.

Μετρικό σφάλμα και λανθασμένες προβλέψεις. Είναι φανερό όμως, πως μια μικρή απομάκρυνση των τερματικών από την πραγματική τους θέση δεν μειώνει κατά πολύ την ποιότητά των προβλέψεων. Συνεπώς, μπορούμε να θεωρήσουμε ότι ένα υποσύνολο $\hat{T}$ είναι χρήσιμο και άρα να λάβουμε υπόψιν την απόστασή του D από τις πραγματικές, ενώ οι υπόλοιπες προβλέψεις Δ είναι λανθασμένες και πρέπει να τις αγνοήσουμε.

Για τη χρήση του αλγορίθμου [Azar et al., 2021] χρειαζόμαστε δύο αλγορίθμους. Ο ένας είναι ο άπληστος άμεσος αλγόριθμος [Imase and Waxman, 1991] ο οποίος είναι $O(\log |R| + 2)$-προσεγγιστικός. Ο δεύτερος είναι ο prize-collecting αλγόριθμος [Goemans and Williamson, 1995] ο οποίος είναι 2-προσεγγιστικός.

Ορίζοντας το μετρικό σφάλμα D ως ένα matching ελάχιστου κόστους ανάμεσα στις καλές προβλέψεις και την είσοδο, έχουμε το ακόλουθο.

Θεώρημα 1.4.5. *[Azar et al., 2021] Το κόστος του αλγορίθμου είναι* $ALG \leq O(\log \min\{\Delta, |R|\} + 2) \cdot OPT + O(1) \cdot OPT$.

Όπως φαίνεται, αυτό το αποτέλεσμα αποτελεί μια σαφή βελτίωση σε σχέση με το προηγούμενο, καθώς χειροτερεύει πιο ομαλά όσο απομακρύνονται οι προβλέψεις.

1.4.4 Άμεσο Facility Location

Το Facility Location είναι ένα ακόμα γνωστό πρόβλημα δικτύων, που ως σκοπό έχει την ομαδοποίηση της εισόδου, ανοίγοντας facilities τα οποία κοστίζουν και πρέπει να βρίσκονται όσο πιο κοντά στους πελάτες που καλύπτει. Πιο συγκεκριμένα, στο μετρικό χώρο $M(X, d)$ έχουμε σύνολα $\mathcal{F}$ με τις δυνατές θέσεις για facilities και $\mathcal{D}$ με τις δυνατές θέσεις των πελατών. Κάθε facility $i \in \mathcal{F}$ κοστίζει f_i και κάθε πελάτης j που ανατίθεται σε αυτό επιφέρει κόστος $d(r_j, f_i)$. Σκοπός είναι να ελαχιστοποιήσουμε το συνολικό κόστος.

Πρόβλεψη της Εισόδου. Παρόμοια με το προηγούμενο πρόβλημα, μπορούμε να χρησιμοποιήσουμε προβλέψεις της εισόδου, οι οποίες δίνονται εξ αρχής.

Για να εφαρμόσουμε τον αλγόριθμο [Azar et al., 2021], χρειαζόμαστε δύο αλγορίθμους. Ο άμεσος αλγόριθμος θα είναι ο primal-dual [Fotakis, 2011] ο οποίος είναι subset-competitive αν θεωρήσουμε ως κόστος του το άθροισμα των $a(r_i)$ που υπολογίζει ο αλγόριθμος και φράσουν το πραγματικό κόστος. Ο λόγος προσέγγισής του είναι $O(\log n)$. Ο prize-collecting αλγόριθμος είναι ο [Xu and Xu, 2005] με λόγο προσέγγισης 1.8526.

Αν το μετρικό σφάλμα D είναι και σε αυτή την περίπτωση ένα matching ελάχιστου κόστους ανάμεσα στις προβλέψεις και την είσοδο έχουμε το εξής.

Θεώρημα 1.4.6. *[Azar et al., 2021] Το κόστος του αλγορίθμου είναι* $ALG \leq O(\log (\min\{|R|, \Delta\} + 2)) \cdot OPT + O(1) \cdot D$.

Πολλαπλές Προβλέψεις της Λύσης. Μία διαφορετική προσέγγιση είναι αυτή των [Almanza et al., 2021], οι οποίοι χρησιμοποιούν προβλέψεις για τις θέσεις των facilities. Έστω δηλαδή ότι έχουμε $S_1, S_2, \ldots, S_k$ σύνολα από facilities. Σκοπός είναι να πετύχουμε μια λύση που συγκρίνεται με τον καλύτερο συνδυασμό facilities από την ένωση των προβλέψεων $S = \bigcup_i S_i$. Έπειτα, μπορούμε να συνδυάσουμε τον αλγόριθμο αυτό με τον βέλτιστο [Fotakis, 2011] χρησιμοποιώντας το αποτέλεσμα των [Mahdian et al., 2012].

Η βασική τεχνική που χρησιμοποιεί ο αλγόριθμος είναι η κατασκευή HST-family $(\mathcal{T}, D)$ που διατηρεί τις ιδιότητες του μετρικού χώρου S των προβλέψεων, σύμφωνα με [Fakcharoenphol et al., 2004].

Ο αλγόριθμος επιλέγει ένα HST σύμφωνα με την κατανομή D. Σε κάθε βήμα, κάνει ένα πέρασμα σε αυτό το δέντρο έτσι ώστε να βρει το φύλλο (facility) στο οποίο θα ανατεθεί ο πελάτης της εισόδου. Στην περίπτωση που δεν υπάρχει κανένα ανοιχτό φύλλο κοντά στον πελάτη, θα χρειαστεί να ανοίξουμε ένα καινούργιο, οπότε θα επιλέξουμε να συνεχίσουμε στο υποδέντρο με τις περισσότερες κοντινές προβλέψεις. Στην αντίθετη περίπτωση θα πρέπει να προσέξουμε κατά πόσο μας συμφέρει να ανοίξουμε κάποιο φύλλο ακόμα πιο κοντά στον πελάτη, ενώ την ίδια στιγμή να αποφύγουμε να ανοίξουμε υπερβολικά πολλά. Για να το πετύχουμε αυτό καθυστερούμε το άνοιγμα μαζεύοντας κόστος σε κάθε κόμβο και προχωρώντας προς το υποδέντρο του αν το κόστος ξεπεράσει το f.

Θεώρημα 1.4.7. *[Almanza et al., 2021] Συνολικά ο αλγόριθμος πετυχαίνει κόστος* $ALG \leq \min\{O(\log |S|) \cdot OPT(S), \frac{\log n}{\log \log n} \cdot OPT\}$.

1.5 Facility Leasing με Προβλέψεις

Το Facility Leasing αποτελεί γενίκευση του Facility Location, με τη διαφορά ότι σε κάθε χρονική στιγμή αλλάζουν οι θέσεις των πελατών και για να τους καλύψουμε νοικιάζουμε facilities για συγκεκριμένη χρονική διάρκεια. Με σκοπό να αναδείξουμε την ισχύ του αλγορίθμου [Azar et al., 2021], το χρησιμοποιούμε στο πρόβλημα Facility Leasing του οποίου η είσοδος κατανέμεται και στο χρόνο εκτός από τον χώρο. Θα χρειαστούμε έναν άμεσο αλγόριθμο που είναι subset competitive καθώς και έναν αλγόριθμο για την prize collecting εκδοχή του προβλήματος. Έπειτα, ορίζοντας κατάλληλο σφάλμα θα βρούμε το λόγο προσέγγισης.

Online Facility Leasing Η είσοδος του προβλήματος χωρίζεται σε T χρονικές στιγμές, άγνωστες κατά την εκτέλεση. Το σύνολο F περιέχει τις δυνατές τοποθεσίες που μπορούμε να νοικιάσουμε facilities, ενώ το D είναι οι δυνατές θέσεις των πελατών. Οι διαφορετικές διάρκειες ενοικίασης είναι K και κάθε μία συμβολίζεται l_k, οπότε αν νοικιάσουμε το f_i τη στιγμή t με το k ενοίκιο τότε θα είναι ανοιχτό στη διάρκεια $[t, t + l_k]$ και θα κοστίσει f_i^k. Κάθε στιγμή εμφανίζεται στην είσοδο ένα σύνολο D_t με τις θέσεις των πελατών r_{jt} τη συγκεκριμένη στιγμή, οι οποίοι πρέπει να ανατεθούν σε κάποιο νοικιασμένο facility f_i με κόστος ανάθεσης την απόσταση $d(r_{jt}, f_i)$. Άν $\mathcal{F}, \mathcal{F}_t$ είναι τα νοικιασμένα facilities και τα ενεργά τη στιγμή t αντίστοιχα τότε το συνολικό κόστος της λύσης είναι

$$\sum_{(i,t,k)\in\mathcal{F}} f_i^k + \sum_{(j,t):r_{jt}\in D_t} d(\mathcal{F}_t, r_{jt})$$

Άμεσος Αλγόριθμος. Ο άμεσος αλγόριθμος για το Online Facility Leasing δίνεται από τους [Nagarajan and Williamson, 2013]. Καθώς φτάνουν οι θέσεις των πελατών την κάθε στιγμή, ο αλγόριθμος προσαρμόζει τις μεταβλητές του δυϊκού προβλήματος, οι οποίες υποδεικνύουν πόσο είναι διατεθειμένος να πληρώσει ο κάθε πελάτης για την ανάθεσή του, έτσι ώστε να διατηρεί την ιδιότητα ότι κανείς δεν ξοδεύει περισσότερο από όσο χρειάζεται. Η κεντρική απόφαση που πρέπει να λάβει ο αλγόριθμος είναι κατά πόσο θα αναθέσει έναν πελάτη σε ήδη νοικιασμένο facility ή αν θα νοικιάσει κάποιο εκ νέου. Αυτό το αποφασίζει αυξάνοντας την μεταβλητή που αντιστοιχεί στον πελάτη μέχρι είτε να καλύψει το κόστος για ανάθεση σε ήδη νοικιασμένο facility ή να συνεισφέρει επαρκώς στην ενοικίαση καινούργιου.

Το παρακάτω λήμμα δείχνει πως ο αλγόριθμος είναι subset competitive εάν στρογγυλοποιήσουμε το κόστος του χρησιμοποιώντας τις μεταβλητές a_{jt} του δυϊκού.

Λήμμα 1.5.1. *Το στρογγυλοποιημένο κόστος του αλγορίθμου για κάθε* $R' \subseteq R$ *είναι*

$$(K+1) \sum_{(j,t)\in R'} a_{jt} \leq 2(K+1)(\log(|R'|+1)+1) \cdot OPT$$

Prize-collecting Αλγόριθμος. Στην περίπτωση της prize-collecting εκδοχής του προβλήματος, μπορούμε να επιλέξουμε να μην αναθέσουμε κάποιους πελάτες σε κανένα facility. Σε αυτή την περίπτωση πληρώνουμε ένα πρόστιμο $\pi(r)$, γνωστό εξ αρχής, για τον πελάτη r.

Λήμμα 1.5.2. *Υπάρχει prize-collecting αλγόριθμος για το Facility Leasing με σταθερό λόγο προσέγγισης ίσο με 3.*

Σφάλμα πρόβλεψης. Για την ανάλυση του αλγορίθμου θα χρειαστεί να βρούμε κάποιον τρόπο να εκφράσουμε το πώς επηρεάζεται από τις προβλέψεις και έτσι να φράξουμε το κόστος του. Στην πραγματικότητα, υπάρχουν πολλοί τρόποι να γίνει αυτό, οπότε ιδανικά θα θέλαμε να ορίσουμε ένα κατάλληλο σφάλμα, που περιγράφει όσο γίνεται καλύτερα το επιπλέον κόστος το οποίο επιφέρουν οι λανθασμένες προβλέψεις. Για αυτόν τον ορισμό θα διατηρήσουμε τη λογική της αντιστοίχισης ένα προς ένα των προβλέψεων με τους πελάτες στην είσοδο, που είναι ικανοποιητικά κοντά. Όπως είναι φανερό, το min cost matching που χρησιμοποιείται για τα υπόλοποιπα προβλήματα, Steiner Tree και Facility Location, δεν επαρκεί καθώς μια οποιαδήποτε λύση που καλύπτει τις προβλέψεις δεν είναι σίγουρο πως μπορεί να χρησιμοποιηθεί για την είσοδο αλλάζοντας τις αναθέσεις. Στην πραγματικότητα, μία πρόβλεψη μπορεί να βρίσκεται χρονικά πολύ μακριά από τους πραγματικούς πελάτες, αλλά να δίνει αρκετή πληροφορία για την κατανομή τους.

Πιο συγκεκριμένα, θα θέλαμε να ορίσουμε το επιλέον κόστος για να καλύψουμε ένα στιγμιότυπο $\hat{T} \subseteq \hat{R}$, εάν έχουμε μια λύση για ένα στιγμιότυπο $T \subseteq R$. Για να βρούμε τα δύο υποσύνολα αρκεί να εφαρμόσουμε τα παρακάτω σε κάθε δυνατό υποσύνολο του R, ενώ τα υπόλοιπα τα υπολογίζουμε στο Δ. Αυτό το επιπλέον κόστος θα βρεθεί κατασκευάζοντας ένα διμερή γράφο $(T, \hat{T})$ με ακμές μήκους ίσου με την απόσταση στο μετρικό χώρο. Όμως θα πρέπει να επιτρέψουμε μόνο μερικές από τις ακμές, με τον παρακάτω τρόπο.

- Αν μια πρόβλεψη $\hat{r}_j \in \hat{R}$ μπορεί να καλυφθεί από κάποιο facility που νοικιάστηκε για το T (και άρα είναι ενεργό τη στιγμή που φτάνει το $\hat{r}_j$), τότε το συνδέω στο διμερή γράφο με όλους τους πελάτες της εισόδου που καλύπτονται από το συγκεκριμένο facility.

Έπειτα σε αυτό τον γράφο λύνουμε το min cost matching και βρίσκουμε αυτό το υποσύνολο T για το οποίο το κόστος που προκύπτει φράσσεται από το D που έχουμε επιλέξει. Μία αδυναμία της παραπάνω διαδικασίας είναι ότι αποτυγχάνει αν η είσοδος έχει σημαντικά διαφορετική διάρκεια από τις προβλέψεις, με την έννοια ότι δεν κάνει καμία αντιστοίχιση αφού η λύση είναι αδύνατο να χρησιμοποιθεί. Αυτό το πρόβλημα λύνεται χρησιμοποιώντας επεκτάσεις των ενοικίων και περιγράφεται με λεπτομέρεια στο αγγλικό μέρος.

Θεώρημα 1.5.1. *Ο αλγόριθμος για το Online Facility Leasing επαυξημένος με προβλέψεις πετυχαίνει το εξής κόστος, για κάθε εφικτή δυάδα* (Δ, D)

$$Alg(R, \hat{R}) \leq O(K) \cdot D + O(K(\log(min\{|R|, \Delta\} + 1) + 1)) \cdot OPT$$

Chapter 2

Introduction

In the study of online algorithms, the main challenge is our uncertainty towards future input. The classical approach has been to analyze the algorithms in comparison with an optimal solution of a given input, in a worst case manner. That is, assuming an adversary is intentionally trying to force the algorithm to make bad decisions. While randomization seems to be a reasonable solution to this problem, in most cases the performance does not improve drastically over that that of deterministic algorithms, which can be obtained through derandomization [Raghavan, 1988]. As a result, the approximation ratios we get in many problems are rather pessimistic. For instance, [Fiat et al., 1991] gives optimal $O(k)$ and $O(\log k)$-competitive deterministic and randomized algorithms for Online Paging, [Imase and Waxman, 1991] gives an optimal $O(\log n)$-competitive greedy algorithm for Online Steiner Tree, [Fotakis, 2011] gives an optimal $O(\frac{\log n}{\log \log n})$-competitive deterministic algorithm for Online Facility Location, [Nagarajan and Williamson, 2013] gives an $O(K \log n)$-competitive algorithm for Online Facility Leasing.

2.1 Previous Work

A novel approach to overcoming uncertainty has been to predict elements of the problem that would assist in achieving better results, as suggested in [Lykouris and Vassilvitskii, 2018], which formalised the study of *learning-augmented algorithms*. The success of machine learning models in a vast range of topics provides enough motivation towards pursuing this goal, as previous instances having appeared in a given application can be used to improve the performance of an algorithm in the future. But even these models are not expected to perform optimally in all scenarios, therefore it is necessary to design algorithms in an appropriate way, so that they take advantage of the predictions as much as they can, while also degrading gracefully towards the best worst-case ratios known, when these prove to be untrustworthy. This is typically achieved by appropriately combining the best known online algorithms with algorithms that blindly trust the predictions, so as to get the best of both worlds. Therefore it is evident that an essential element in the analysis of such algorithms will be the notion of an *error metric* between what is predicted and what is actually given from the input. Depending on the problem, this error metric can be a simple L_1 norm between the items bought by a possible solution, or as complicated as a the solution of a combinatorial optimisation problem in itself. An obvious challenge is that since the input is unknown, the quality of the predictions is impossible to evaluate beforehand, meaning that the algorithms will need to balance between the two in an online fashion. Another question that naturally arises is *what should we predict*? There are cases where the prediction is chosen based on the future information that an optimal offline algorithm would take advantage of. For example, in the case of Online Paging in [Lykouris and Vassilvitskii, 2018], it is Belady's algorithm [Belady, 1966] that hints towards predicting the next arrival of each page. Some authors have attempted to use predictions of the whole input [Xu and Moseley, 2021, Azar et al., 2021] while others used predictions on the optimal solution [Bamas et al., 2020, Fotakis

et al., 2021, Jiang et al., 2021, Agrawal et al., 2022, Balkanski et al., 2022]. Motivated by learning theory, some works have been successful in utilising multiple predictions in order to achieve even better results [Almanza et al., 2021, Anand et al., 2022a].

The area of online algorithms has certainly been the one with the most success in utilising predictions. Starting with the seminal work on Online Paging [Lykouris and Vassilvitskii, 2018], it was later improved by [Rohatgi, 2019, Wei, 2020] and [Antoniadis et al., 2020], who used a different error metric. Its generalisation, Online Weighted Paging was studied in [Jiang et al., 2020], proving the limitation of predicting only the subsequent time each page will be called and using more predicted information, while [Bansal et al., 2020] gave optimal randomized and deterministic algorithms matching the previous bounds. Another classic problem studied in this context is Non-Clairvoyant Scheduling, originally in the single-machine unweighted version [Purohit et al., 2018, Mitzenmacher, 2019, Im et al., 2021a], single-machine weighted version [Cho et al., 2022], but also in more general settings with multiple machines and weighted jobs [Lindermayr and Megow, 2022, Zhao et al., 2022]. Moreover, network problems like Online Steiner Tree [Xu and Moseley, 2021] and Online Facility Location [Almanza et al., 2021, Fotakis et al., 2021, Jiang et al., 2021] have been studied, with more general results in [Azar et al., 2021, Bernardini et al., 2022]. Some other classic problems set in this context are Online Subset Sum [Xu and Zhang, 2022], Online Knapsack [Thang and Durr, 2021, Im et al., 2021b, Boyar et al., 2022]. As for the optimality of consistency-robustness tradeoffs, it is studied for Ski Rental and Online Paging in [Wei and Zhang, 2020].

Having said that, the applications are not restricted to online algorithms, but a variety of problems, from mechanism design [Agrawal et al., 2022, Balkanski et al., 2022, Gkatzelis et al., 2022, Istrate and Bonchis, 2022, Xu and Lu, 2022], to improving the running time of classic algorithms [Lu et al., 2020, Polak and Zub, 2022, Dinitz et al., 2021, Feijen and Schäfer, 2021]. Their success can be evidenced both by the theoretical guarrantees provided as well as the experiments designed to test their strength.

So far, most of the work in this area focuses on ways to leverage the predictions, assuming that they are provided by an oracle. On the other hand, it is natural to ask whether the predictions we use can be *learnt* so that they are of high quality and what the sample complexity is in each case. This is the direction pursued by [Du et al., 2021, Khodak et al., 2022, Anand et al., 2022c, Anand et al., 2022b].

In addition to problem specific methods, general frameworks have also been developed [Bamas et al., 2020, Anand et al., 2022a, Azar et al., 2021], some of which could potentially provide black-box solutions for different categories of problems. In [Bamas et al., 2020], the primal-dual method is extended to the learning-augmented scenario and used in order to obtain algorithms for Online Weighted Set Cover, Ski Rental and TCP Acknowledgment, when given predictions of the solution. Next, [Anand et al., 2022a] deals with the case of multiple predictions supplied in an online fashion to the variables of a linear program of a covering problem, like Online Facility Location. On the other hand, [Azar et al., 2021] was designed specifically for metric problems, like Online Steiner Tree and Online Facility Location, given a prediction of the whole input in an offline fashion. The new notion of error it introduces seems to capture the characteristics of these problems, while also being general enough to adapt to many different problems. Recently, a universal error metric was found for this specific framework which can be used to show that the actual performance is even better in some cases [Bernardini et al., 2022]. Therefore, these frameworks cover a large amount of problems, as well as various different choices for the predictions.

2.2 Facility Leasing

We proceed to obtain a learning-augmented algorithm for a problem that had not been previously considered in this study, Online Facility Leasing, whose offline counterpart first appeared in [Anthony and Gupta, 2007]. In this variant of Facility Location, the locations of the demands change on every time step and in order to cover them we lease facilities, choosing from K available lease periods. Therefore, it can be seen as a generalization of both the Online Facility Location [Meyerson, 2001] and the Parking Permit problem [Meyerson, 2005]. Intuitively, this problem could model the distribution of clients of a company as it changes every year. In this sense it is a more realistic depiction of actual decision making, where one seeks to lease goods for only a limited amount of time, in order to adapt to the changes of the environment.

2.2.1 Framework

Surprisingly, the framework [Azar et al., 2021] designed for network problems, is sufficient for a greater family of problems, as Facility Leasing combines network properties with a time dependency. In this framework, the input is predicted, providing a whole instance to the algorithm in an offline manner. The key is that an online algorithm responsible for covering the online input, is combined with an offline one responsible for buying useful solutions covering some of the offline predictions. In fact, the online algorithm is allowed to run for exponential steps of the online cost, at which points the offline algorithm injects solutions from the predicted instance, covering as many predictions as possible with a cost comparable to that of the online algorithm, hoping that they will prove useful in the future. For the analysis of this algorithm, they introduce a new notion of error, that of a tuple (Δ, D). In this tuple, Δ measures how many predictions were entirely wrong, simply called outliers, while D measures a distance between the input and the correct predictions, in terms of their distributions. The intuition behind the error is that the offline algorithm will buy solutions for the correct predictions, which the online algorithm will later connect to the actual input, while the outliers will be ignored completely.

2.2.2 Algorithms

The online algorithm is given by [Nagarajan and Williamson, 2013], which modifies the primal dual algorithm of [Fotakis, 2011] and employs a dual-fitting analysis. In essence, it increases the dual variable of the new demand and by finding which dual constraints become tight, it decides whether a lease has been paid and should be opened, or whether the new demand should be assigned to an already leased facility. The algorithm achieves a $K \log n$ competitive ratio. It is not known whether this is tight, nor optimal, as it is the only algorithm known for this problem.

The offline algorithm [de Lima et al., 2016] solves the prize-collecting version of the problem, so that it can buy solutions for some of the predictions. This algorithm uniformly increases the dual variables until some dual constraint is violated, keeping in mind that there is an extra set of constraints due to the penalties. By carefully selecting which of the facilities to lease and tripling them, it achieves a 3-approximation.

2.2.3 Contribution

Following this work, we defined an appropriate notion of distance between instances of Facility Leasing. The difference is that on top of a spacial distribution, the demands of Facility Leasing are also distributed in time, both of which need to be considered in order to define a

reasonable matching cost between the two instances. That implies that a demand arriving at time t cannot be matched with any prediction, as a solution leased to cover that prediction might not be open at t. Instead, it can only be matched with the predictions whose leased facilities are open at that time. But even then, should correct predictions that appear too far in the future be excluded? The answer is no, as they can still provide useful insight, but need to be assigned a higher matching cost, that would pay for extending the lease of a facility so that it might cover both. In essence, lease extensions allow us to measure the time cost between the two distributions.

With that in mind, we obtain an algorithm with approximation ratio upper bounded by $O(K(\log(|\Delta|)+1)) \cdot OPT + O(K) \cdot D$. Supposing that the predictions are good enough, Δ is 0, while D can be thought of as an ϵ, in which case the approximation ratio is K, much better than that of the best worst case online algorithm. On the opposite case, there is no sense in trying to match the demands and $D = 0$, leading to a ratio of $O(K(\log(n)+1)) \cdot OPT$ which matches the best known online algorithm. Overall, it is evident that noticeable improvements can be achieved by using these novel techniques in Facility Leasing, without compromising beyond a constant factor in the approximation ratio.

Chapter 3

Preliminaries

We proceed by exploring different ways in which predictions are incorporated in the design of algorithms in order to exceed the performance of the state of the art worst case algorithms. Despite the fact that these techniques only appeared recently, many problems have been studied, namely

- Ski Rental
- Job Scheduling
- Paging
- Steiner Tree
- Facility Location

In order to begin our study of learning augmented algorithms some basic notions are needed to facilitate the analysis.

3.1 Definitions

In their seminal work [Lykouris and Vassilvitskii, 2018], Lykouris and Vassilvitskii set the framework for the analysis of learning augmented algorithms, by extending the idea of the approximation ratio to accommodate the characteristics of learning augmented algorithms.

Definition 3.1.1 (Approximation Ratio)**.** *An algorithm A has approximation ratio c for all instances σ of a minimization problem if*

$$cost_A(\sigma) \leq c \cdot OPT(\sigma)$$

Essentially, a learning augmented algorithm should provide guarrantees that it gets close to the optimal solution when the predictions, whatever they might be, are perfect. It should also degrade gracefully to the performance of the best worst case algorithm even when the predictions are not representative of the input. These ideas are expressed by the *consistency* and the *robustness* of the algorithms as defined below, in a rather simplified manner.

Definition 3.1.2 (Consistency)**.** *An algorithm is considered* β-consistent *if its approximation ratio is β when the predictions' error is minimum, i.e. the predictions are correct.*

Definition 3.1.3 (Robustness)**.** *An algorithm is considered* α-robust *if its approximation ratio is not worse than α for any possible error of the predictions.*

Using the above definitions, for an online learning augmented algorithm to be considered effective, it is expected to achieve a consistency much better than that of its worst case

counterpart, preferrably constant, and a robustness that asymptotically matches that of the worst case.

As for the algorithms encountered, a crucial property in the framework [Azar et al., 2021] is that of *subset competitiveness*. Intuitively, an algorithm is subset competitive if the solutions it produces preserve their competitive ratio on all subsets of the solution against the optimal solution on that subset.

Definition 3.1.4 (Subset Competitive)**.** *An online algorithm ON input R and approximation ratio $f(|R|)$ is* subset competitive *when for every $R' \subseteq R$ the cost incurred by the solution of ON given R on the elements of R' is*

$$ON(R') \leq f(|R'|) \cdot OPT(R')$$

This property is useful when considering the costs incurred on parts of the input with useful properties, especially when combining multiple algorithms and keeping track of each one's contribution to the cost.

We now turn to a category of variations of problems called *prize-collecting* problems. In essence, given an online input that needs to be covered by making irrevocable decisions, prize-collecting variations relax this requirement by providing an alternative. Either cover the element given by paying the cost of an appropriate solution, or pay a penalty and discard it.

Definition 3.1.5 (Prize-collecting)**.** *Suppose that given an online input R for a problem, the solution pays $c(r)$ for an $r \in R$ if it chooses to satisfy it, where c is dependent on the structure of the problem, or $\pi(r)$ and not satisfy it, where $\pi : R \to \mathbb{R}^+$ is a penalty function given as offline input. The goal is to minimize the objective*

$$\sum_{r \in S} c(r) + \sum_{r \in R \setminus S} \pi(r)$$

where $S \subseteq R$ is the set of satisfied requests $r \in R$.

This category of problems is particularly useful when trying to obtain a solution that only partially covers the input. By assigning different values to the penalties $\pi(e)$ and inspecting the resulting solution, it is possible to find a part of the input of desired size, which can be covered incurring a low cost. Intuitively, high penalties incentivize satisfying a larger part of the input in order to avoid the high cost, while lower penalties lead to smaller portions of the input being satisfied.

3.2 Predictions

Depending on the problem, there might be different elements needed by an online algorithm to make up for the uncertainty of the input. In general, there are two natural choices made by the majority of the works in this field. The first one is to predict what we want to achieve, that is the actual solution or parts of it, that will satisfy the future input. The second one is to predict the one thing that restricts us in the study of online algorithms, that is the input. Another approach, is to make multiple predictions in the hopes that some of them will be good.

Definition 3.2.1 (Predictor)**.** *Let $\mathcal{X}$ be the set of possible instances for a problem and $\mathcal{Y}$ a set of labels. Then the predictor is a function $h : \mathcal{X} \to \mathcal{Y}$. Specifically, for every possible instance $\sigma \in \mathcal{X}$ it outputs the label $h(\sigma)$.*

In the above definition we do not specify what these instances might be, but in most cases they consist of the offline input of the problem i.e. the metric space in which it is defined, and maybe the online input.

The various kinds of predictions will be discussed below.

Ski Rental. A seemingly simple problem called *Ski Rental*, where the main decision on each day is whether to rent the skis or buy them for a fixed cost provides enough motivation for this approach. The prediction proposed in [Purohit et al., 2018] is the total amount of days that the person stays in the resort. Expanding upon this result, [Gollapudi and Panigrahi, 2019] assume k predictions of the total amount of days.

Paging. Three models of predictions have been used for the study of *Online Paging*. The first is the Per-Request Prediction model, inspired by the Belady's rule [Belady, 1966], which provides for each requested page the subsequent time when it will be requested and is used for the unweighted version by [Lykouris and Vassilvitskii, 2018, Rohatgi, 2019, Wei, 2020] and for the weighted version by [Bansal et al., 2020]. On the other hand, the Strong Per-Request Prediction model used by [Jiang et al., 2020], obtains better results by demanding that the predictions also give the whole sequence of page requests until the next request of the same page. Finally, in [Antoniadis et al., 2020] a different model is used, inspired by the study of Metrical Task Systems, which predicts the state of the whole cache for every time step and can be derived from the Per-Request Prediction model.

Job Scheduling. The study of *Non-Clairvoyant Job Scheduling* in the context of predictions started with [Purohit et al., 2018], which studied the unweighted version with one machine with predictions being the processing time of each job and later improved by [Im et al., 2021a]. A similar choice is made in the more general case of multiple jobs and different machines in [Zhao et al., 2022], while [Lindermayr and Megow, 2022] predict the whole permutation of jobs that will be assigned to each machine, i.e. the optimal solution. On the other hand [Cho et al., 2022] studied the case where there is a single machine and weighted jobs and predicted the weight of each job. From a mechanism design perspective, [Balkanski et al., 2022] naturally predicts the private information of each agent, which is the processing load that the job would incur to their machine.

Steiner Tree. As for network problems, the *Online Steiner Tree* has been studied extensively. In a completely different setting compared to the previous problems, both [Xu and Moseley, 2021, Azar et al., 2021] consider a model in which the predictions are the terminals, provided in an offline fashion to the algorithm.

Facility Location. Another interesting network problem, *Online Facility Location* was studied in order to utilise predictions. In [Azar et al., 2021] the predictions are given in an offline fashion and correspond to the locations of all the demands. On the other hand, a different approach is employed by [Fotakis et al., 2021, Jiang et al., 2021], where each demand arriving online is given a corresponding prediction as to which facility it should be assigned to. Furthermore, cast in the setting of mechanism design, this problem was studied by [Agrawal et al., 2022], where a single prediction was requested, the location of the optimal centre for the facility.

3.3 Error Metrics

In order to analyze an algorithm that utilises predictions, it is necessary to define appropriate notions of error for these predictions. Informally, this error can be thought of as the extra cost incurred for extending a solution bought according to the predictions, in order to cover the actual input. This way, we can achieve bounds parametrized by this error, thus beating the worst case. At this point it is important to note that the analysis of learning augmented

algorithms does not require any guarrantees from these predictions. As a result, the bounds given, hold for any possible value of the error. This is very useful as it provides the necessary separation between the oracle that offers predictions and the algorithm that uses them.

Next, we explore some of the errors that have been used in the literature, for each of the problems.

Ski Rental. In the case of a single prediction, [Purohit et al., 2018] uses the difference $\eta = |\hat{x} - x|$ between the predicted days $\hat{x}$ and the actual days x, which can be thought of as a trivial use of the L_1-norm. On the other hand, when multiple predictions are considered, [Gollapudi and Panigrahi, 2019] use the difference between the best prediction and the actual days of skiing, $\eta = \min_i |\hat{x_i} - x|$.

Paging. In online caching, [Lykouris and Vassilvitskii, 2018] define the error in a more general manner, for any loss function l with arguments the actual next time that an element σ_i will be requested, $y(\sigma_i)$, and the predicted next time, $h(\sigma_i)$, as $\eta = \sum_i l(y(\sigma_i), h(\sigma_i))$. Examples that could be used for l are the absolute difference, which naturally leads to η being the absolute difference and the squared difference which leads to η being the squared loss function. Similarly, [Rohatgi, 2019,Wei, 2020] use as error the sum of absolute differences, but obtain bounds using the number of inversions, i.e. the number of pairs of pages whose next requests are predicted in the opposite order, which is upper bounded by twice the error. As a generalisation, in the weighted version [Bansal et al., 2020, Jiang et al., 2020] use the weighted absolute difference as an error metric. On the other hand, [Antoniadis et al., 2020] approaches the problem as an MTS and therefore sums the distances between the predicted and optimal cache states at each timestep.

Job Scheduling. In *Non-Clairvoyant Job Scheduling*, with predictions $\hat{x}_i$ of the total processing times x_i, the error was used in [Purohit et al., 2018] as the L_1-norm $\eta = \sum_i |\hat{x}_i - x_i|$. This was refined in [Im et al., 2021a] where a more elaborate definition was employed, using the increase in the cost of an optimal solution if overrestimated (also underrestimated) predictions were correct. On the other hand in [Lindermayr and Megow, 2022], where permutations are predicted, the error measures the increase in the cost incurred by inverting pairs of jobs. In [Zhao et al., 2022], the error is simply the maximum ratio between predicted and actual size of a job. Surprisingly, a similar error is used in the context of mechanism design, where [Balkanski et al., 2022] use the maximum ratio between predicted actual processing cost for each job, agent.

Steiner Tree. As for the Steiner Tree, two different models have been proposed. In the framework by Moseley [Xu and Moseley, 2021], the error was defined as the number of terminals not accurately predicted, despite the fact that their distance to the actual terminals might have been minuscule. On the other hand, in the framework [Azar et al., 2021] the error incorporates both the notion of *outliers* and *matched* predictions, with the tuple (Δ, D). The matching cost is chosen as the distance (cost of the shortest path) between each prediction and terminal in the input on the graph, which are then matched by a min cost matching so that the total cost does not exceed D. As expected, the analysis of the algorithm is valid for any feasible such tuple. In that sense it generalises the former approach, which can be seen by setting the matching cost $D = 0$, in which case only predictions that are located on actual terminals are matched.

Facility Location. First, for the case where the positions of the facilites are predicted, the error used in [Fotakis et al., 2021, Jiang et al., 2021] is mainly the L_∞-norm between the predictions and the actual optimal centers, $\eta = \max_i d(f_{x_i}^{pred}, f_{x_i}^{opt})$. One technical detail, is that in order for this error to be bounded, the predictions are calibrated appropriately. Second, the case where the whole input is predicted is identical to that of Steiner Tree. In the mechanism design variation, [Agrawal et al., 2022] normalise the L_1-norm of the predicted and

optimal location of the facility, $\hat{c}$ and c respectively, by the optimal social welfare, $\eta = \frac{d(\hat{c},c)}{OPT}$.

All these applications provide some insight into what is effective in the choice of predictions and errors, which are undoubtedly the crux of this field of algorithms. While there is no consensus as to what is suitable for each problem, some conclusions can be made. Problems where a characteristic of each element of the input is desirable, admit the obvious choice, a norm that aggregates the individual differences between each prediction and each actual characteristic. On the other hand, when we wish to use a prediction of the whole input, it is reasonable to be more interested in the differences in terms of the distribution between the predictions and the input, especially in network problems, but not exclusively. Accounting for the fact that some predictions might be faulty, we can expand on the previous notion by discarding some predictions as outliers, which are expected to be present in realistic scenarios and should not be allowed to skew the analysis towards unbounded costs. Last of all, another useful idea is that of normalising the error by the cost of the optimal solution, in order to prevent it from increasing with size of the input.

Chapter 4

Frameworks

Even though the study of learning augmented algorithms is still in an exploratory phase, where most problems are approached by ad hoc methods, certain frameworks have emerged, which facilitate the design of such algorithms for families of problems. These frameworks utilise ideas from the study of online algorithms, like the primal dual method, the multiple experts problem or the online combination of online algorithms, in order to provide a more general approach to taking advantage of predictions. The results that will be presented have been developed with different kinds of covering problems in mind. One interesting question is whether they can be adapted to solving more general problems when predictions are available.

4.1 Primal Dual Learning Augmented Framework

The primal dual method for online algorithms appeared for the first time in [Alon et al., 2003], in order to solve the fractional weighted set cover problem. Its main property is that it maintains a feasible solution to the primal LP, while also maintaining an infeasible solution to the dual LP. Conceptually, the following steps are needed to use it.

1. Find the factor a by which the solution of the dual needs to be scaled down to become feasible.

2. Upper bound the feasible primal solution by some factor b times the dual.

3. Using weak duality the product $a \cdot b$ is the approximation ratio.

If we wish to obtain an integral solution, the ratio only increases due to the integrality gap of the problem.

Similarly to the classical primal-dual method, the framework of [Bamas et al., 2020] follows the same steps to prove consistency and robustness. The main idea, is that if we are given a prediction of an integral solution to the problem, we can introduce a tunable parameter λ to increase or decrease our trust in the predictions and how much they affect the updates.

Weighted Set Cover with Predictions. The problem is once again weighted set cover, which is general enough to showcase the properties of the framework. We are given a set $\mathcal{U} = \{e_1, e_2, \ldots, e_n\}$ and a weighted set $\mathcal{F}$ of subsets of U. In the fractional problem, we need to cover every element by fractionally buying x_S of each $S \in \mathcal{F}$ in order to satisfy $\sum_{S \in \mathcal{F}(e)} x_S \geq 1$ while minimizing the total cost $\sum_{S \in \mathcal{F}} w_S \cdot x_S$. In this scenario, we are also provided with a set $\mathcal{A} \subseteq \mathcal{F}$ which predicts the optimal integral solution. Adapting the original primal-dual algorithm, we can tune our trust into the sets that have been predicted to be part of the solution by updating them proportionately.

Algorithm 3: PDLA Weighted Set Cover

forall *e in the input* **do**
  while $\sum_{S \in F(e)} x_S < 1$ **do**
    forall $S \in F(e)$ **do**
      $x_S \leftarrow x_S \cdot (1 + \frac{1}{w_S}) + \frac{\lambda}{w_S |\mathcal{F}(e)|} + \frac{1-\lambda}{w_S |\mathcal{F}(e) \cap \mathcal{A}|} \cdot 1\{S \in \mathcal{A}\}$
    $y_e \leftarrow y_e + 1$

At this point we note that this algorithm does not cover the case when the predictions are not a feasible solution, but this can be solved with minor changes both in the algorithm and in the analysis and can be found in [Bamas et al., 2020].

The analysis of this algorithm will closely follow the 3 steps outlined above, adding another one for the consistency ratio.

Lemma 4.1.1. *The scaled down dual solution with variables* $\frac{y_e}{\Theta(\log(d/\lambda))}$ *is feasible.*

Proof. The dual problem only requires that $\sum_{e \in S} y_e \leq 1$ for every $S \in \mathcal{F}$. In order to obtain a bound we will first notice that every x_S has a lower bound of a geometric series with exponent exactly equal to that sum. Next, we will upper bound this x_S by a constant.

Inspecting the update rule for an x_S that has been updated k times we notice the following recursion, where $x_S^0 = 0$

$$\begin{aligned}
x_S^k &= x_S^{k-1}(1 + \frac{1}{w_S}) + \frac{\lambda}{w_S |\mathcal{F}(e)|} + \frac{1-\lambda}{w_S |\mathcal{F}(e) \cap \mathcal{A}|} \cdot 1\{S \in \mathcal{A}\} \\
&\geq x_S^{k-1}(1 + \frac{1}{w_S}) + \frac{\lambda}{d \cdot w_S} \\
&\geq x_S^{k-2}(1 + \frac{1}{w_S})^2 + \frac{\lambda}{d \cdot w_S}(1 + \frac{1}{w_S}) + \frac{\lambda}{d \cdot w_S}(1 + \frac{1}{w_S})^0 \\
&\vdots \\
&\geq \frac{\lambda}{d \cdot w_S} \sum_{i=0}^{k-1} (1 + \frac{1}{w_S})^i \\
&= \frac{\lambda}{d} \left((1 + \frac{1}{w_S})^k - 1 \right)
\end{aligned}$$

Now, notice that x_S will be updated as many times as $e \in S$ have arrived until a given point. Therefore $k = \sum_{e \in S} y_e$. Using the inequality $(1 + \frac{1}{w_S})^{w_S} \geq 2$ we get

$$\begin{aligned}
x_S^k \geq \frac{\lambda}{d} \left(2^{k/w_S} - 1 \right) \Leftrightarrow \\
\sum_{e \in S} y_e \leq w_s \cdot \log \left(\frac{d}{\lambda} x_S^k + 1 \right)
\end{aligned}$$

What remains is to give the upper bound, noting that $x_S \leq 1$ at all times before the last

$$\begin{aligned}
x_S^k &= x_S^{k-1}(1 + \frac{1}{w_S}) + \frac{\lambda}{w_S |\mathcal{F}(e)|} + \frac{1-\lambda}{w_S |\mathcal{F}(e) \cap \mathcal{A}|} \cdot 1\{S \in \mathcal{A}\} \\
&\leq 2x_S^{k-1} + 1 \\
&\leq 2 \cdot 1 + 1 = 3
\end{aligned}$$

Combining the two bounds we get

$$\sum_{e \in S} y_e \leq w_S \cdot \log\left(\frac{d}{\lambda} \cdot 3 + 1\right)$$

which proves the lemma.

□

Lemma 4.1.2. *The primal solution is feasible and it is upper bounded by* 2 *times the dual.*

Proof. The variables x_S are only increased by the updates. Therefore, once a constraint $\sum_{e \in S} x_S \geq 1$ has been satisfied, which occurs when the while loop ends, no variable in the sum can be decreased.

We proceed to prove the ratio between the primal and the dual solution, by showing that every single increase of a variable y_e in the dual incurs at most twice that in the primal solution. Suppose that an update takes place, after e arrives. Once all x_S with $S \in \mathcal{F}(e)$ have been updated, y_e increases by exactly 1.

Formally, $\Delta D = 1$ while $\Delta P = \sum_{S \in \mathcal{F}(e)} w_S \cdot (x_S^{'} - x_S)$. Denoting $\delta_S = x_S^{'} - x_S$ we have two different cases.

- If $S \in \mathcal{F}(e) \cap \mathcal{A}$ then
$$w_S \cdot \delta_S = x_S + \frac{\lambda}{|\mathcal{F}(e)|} + \frac{1-\lambda}{|\mathcal{F}(e) \cap \mathcal{A}|}$$
- If $S \in \mathcal{F}(e) \setminus \mathcal{A}$ then
$$w_S \cdot \delta_S = x_S + \frac{\lambda}{|\mathcal{F}(e)|}$$

By summing them up we get

$$\Delta P = \sum_{S \in \mathcal{F}(e)} x_S + \lambda \cdot \frac{|\mathcal{F}(e) \cap \mathcal{A}|}{|\mathcal{F}(e)|} + 1 - \lambda + \lambda \cdot \frac{|\mathcal{F}(e) \setminus \mathcal{A}|}{|\mathcal{F}(e)|} = \sum_{S \in \mathcal{F}(e)+1} x_S + 1 < 2 = 2 \cdot \Delta D$$

since before the update, the constraint $\sum_{S \in \mathcal{F}(e)} x_S \geq 1$ was not satisfied. □

At this point, we know that the primal solution we have obtained is feasible, achieving a robustness of $\Theta(\log(d/\lambda))$.

Theorem 4.1.1. *The cost of the algorithm is* $ALG \leq \min\{\Theta(\log(d/\lambda)) \cdot OPT, O(\frac{1}{1-\lambda}) \cdot S(\mathcal{A}, \mathcal{I})\}$*, where* $S(\mathcal{A}, \mathcal{I})$ *is the cost of the predicted solution.*

Proof. The first bound has already been proved. As for the second one, it shows the consistency of the algorithm. In order to prove it, we will have to charge all costs to those incurred by the variables x_S where $S \in \mathcal{A}$. Notice that these are upper bounded by 3, implying that their total cost is at most $3 \cdot S(\mathcal{A}, \mathcal{I})$.

We begin by breaking the primal cost into two parts $\Delta P_c, \Delta P_u$, which describe the increase in cost after an update to the variables whose $S \in \mathcal{F}(e) \cap \mathcal{A}$ and $S \in \mathcal{F}(e) \setminus \mathcal{A}$ respectively. As already shown in the previous lemma,

$$\Delta P_c = \sum_{S \in |\mathcal{F}(e) \cap \mathcal{A}|} x_S + \lambda \cdot \frac{|\mathcal{F}(e) \cap \mathcal{A}|}{|\mathcal{F}(e)|} + 1 - \lambda \geq \frac{\lambda}{d} + 1 - \lambda$$

$$\Delta P_u = \sum_{S \in |\mathcal{F}(e) \setminus \mathcal{A}|} x_S + \lambda \cdot \frac{|\mathcal{F}(e) \setminus \mathcal{A}|}{|\mathcal{F}(e)|} \leq 1 + \lambda$$

Then the ratio between them is $\frac{\Delta P_u}{\Delta P_c} \leq \frac{1+\lambda}{\frac{\lambda}{d}+1-\lambda}$ and the total increase is

$$\Delta P \leq \left(1 + \frac{1+\lambda}{\frac{\lambda}{d}+1-\lambda}\right) \cdot \Delta P_c = \left(\frac{2}{\frac{\lambda}{d}+1-\lambda}\right) = O\left(\frac{1}{1-\lambda}\right) \cdot \Delta P_c$$

By summing all the increases and charging P_c to $S(\mathcal{A},\mathcal{I})$ we have the theorem. □

4.2 Covering Problems with Multiple Predictions

Staying in the area of covering problems we focus on a more general scenario, where we are provided with k different predictions at each step. Specifically, [Anand et al., 2022a] studied the online covering problem, a generalisation of fractional weighted set cover, where in each step k predictions are given in an online fashion and gave a framework for utilising predictions for fractional problems in this category.

Online Covering Problem. The online covering problem (OCP), can be formulated by the following LP, assuming costs $c_i \geq 0$ and constraint coefficients $a_{ij} \geq 0$

$$\begin{aligned} \min & \sum_i c_i \cdot x_i \\ s.t. & \sum_i a_{ij} \cdot x_i \geq 1 \quad \forall j \in [m] \\ & x_i \in [0,1] \quad \forall i \in [n] \end{aligned}$$

The costs c_i are given as offline input. In each step j of the online input, a new constraint $\sum_i a_{ij} \cdot x_i \geq 1$ is given and needs to be satisfied by deciding new values for the variables x_i. One common way of ensuring that no previous constraint will be violated, is to only increase the variables for every new constraint. Since the constraints of the coefficients are not negative, if a constraint is satisfied, increasing any variable will only cause the sum to exceed 1.

Predictions. As mentioned earlier, in each step j we receive k predictions, each one suggesting different values for all variables x_i. The s-th prediction for variable x_i at time j is denoted $\hat{x}_i(j,s)$. These predictions are assumed to all satisfy the j-th constraint and nothing else. In fact, an even stronger assumption is made to simplify the analysis, that $\sum_i c_i \cdot \hat{x}_i(j,s) = 1$, but it is without loss of generality, since even if it exceeded it, we could lower some variable in it. Therefore, their actual values are not expected to be used as is, rather they will hint towards a good solution. To formalise this notion, the following definition is given

Definition 4.2.1. *The value of a variable x_i is said to be* supported *by the s-th prediction if $x_i \geq \hat{x}_i(j,s)$ for all $j \in [m]$.*

Benchmarks. In order to provide consistency bounds for the algorithm, we need to define some notion of *good* solution according to the predictions, which is going to be compared to it.

One natural way to define such a solution is to consider the best individual predictor, out of the k available and find the best solution that is supported by them. This solution will be called $STATIC$. Formally,

$$STATIC = \min_{s \in [k]} \sum_{i \in [n]} c_i \cdot \max_{j \in [m]} \hat{x}_i(j,s)$$

A different approach would be to freely choose the best from all the predictions. The set of available choices will be $\hat{X} = \{\hat{x} \in [0,1]^n \mid \forall i \in [n], \forall j \in [m], \exists s \in [k] : \hat{x}_i \geq \hat{x}_i(j,s)\}$. By this definition, every $\hat{x}_i$ is supported by some predictor for every step, thus allowing each component to be supported by a different predictor. This solution will be called $DYNAMIC$.

$$DYNAMIC = \min_{\hat{x} \in \hat{X}} \sum_{i \in [n]} c_i \cdot \hat{x}_i$$

Obviously the available options of the $DYNAMIC$ solution include those that can be made by the $STATIC$, therefore

$$DYNAMIC \leq STATIC$$

Online Covering Algorithm with Predictions. As in the primal-dual framework, in every step the algorithm will be increasing the variables by some rate $\frac{dx_i}{dt}$. This rate will depend on the cost c_i incurred by the variable, the benefit it offers towards satisfying the current constrain, a_{ij}, as well as the k suggested values for it. In order to incorporate all of the predictions in the calculation of the rate, a natural choice is to use their average $\frac{1}{k}\sum_s \hat{x}_i(j,s)$. In the algorithm and the analysis, the notation $\delta = \frac{1}{k}$ and $x_{ij} = \sum_s \hat{x}_i(j,s)$ will be used. Initially all x_i are equal to 0. At this point we are ready to present the algorithm.

Algorithm 4: Online Covering Framework

```
forall steps j do
    while Σ_{i∈[n]} a_ij x_i < 1/2 do
        forall i ∈ [n] do
            if x_i < 1/2 then
                Increase x_i with rate dx_i/dt = (a_ij / c_i) · (x_i + δ · x_ij)
```

In the end, all constraints will be half satisfied, therefore we can get a feasible solution by multiplying every variable by 2, which increases the cost by the same constant factor. Another property of the algorithm, is that if $\sum_{i \in [n]} a_{ij} x_i < \frac{1}{2}$, then there will always be at least one variable $x_i < \frac{1}{2}$ which will be increased in order to help satisfy the constraint. Otherwise, if all $x_i \geq \frac{1}{2}$ then $\frac{1}{2}\sum_{i \in [n]} a_{ij} \leq \sum_{i \in [n]} a_{ij} x_i < \frac{1}{2}$ and $\sum_{i \in [n]} a_{ij} < 1$, which means that the constraint given is impossible to satisfy by any vector in $[0,1]^n$. As for the continuous rate of increase, it can be discretized, by increasing the variable until either the variable itself or the sum of the constraint exceed $\frac{1}{2}$.

Analysis. The analysis of the algorithm uses the potential method. Specifically, by comparing the rate of increase of the cost incurred by the algorithm to the rate of decrease of the potential function, we acquire an upper bound for the cost. Next, by bounding the potential function by the $DYNAMIC$ solution we obtain the consistency bound $O(\log k \cdot DYNAMIC)$. As for the robustness, a slight modification of the algorithm is used in order to combine the predictions with the solutions suggested by a worst case online algorithm.

Lemma 4.2.1. *The rate at which the cost incurred by the algorithm increases is at most* $\frac{3}{2}$.

Proof. The rate of increase of the total cost is

$$\begin{aligned}\sum_i c_i \cdot \frac{dx_i}{dt} &= \sum_i a_{ij}(x_i + \delta x_{ij}) \\ &= \sum_i a_{ij}x_i + \frac{1}{k}\sum_i \sum_s a_{ij}\hat{x}_i(j,s) \\ &< \frac{1}{2} + \frac{1}{k}\sum_s \sum_i a_{ij}\hat{x}_i(j,s) = \frac{3}{2}\end{aligned}$$

□

In order to upper bound the total cost incurred, we will define a potential function which models the extra cost incurred by the $DYNAMIC$ solution at each step. The potential function for each variable is defined by a component

$$\phi_i = c_i x_i^{DYN} \ln \frac{(1+\delta)x_i^{DYN}}{x_i + \delta x_i^{DYN}}$$

where each x_i has the value it was assigned with by the algorithm at any step, while x_i^{DYN} is constant and is the value of the $DYNAMIC$ solution.

The total potential function is the sum of those components for the variables in which the $DYNAMIC$ solution actually incurred extra cost

$$\phi = \sum_{i:x_i^{DYN} \geq x_i} \phi_i$$

In order for this potential function to be useful in bounding the total cost, we will need for it to have a constant rate of decrease and also for it to be non negative so that we can directly correlate the two rates with the corresponding maximum values.

Lemma 4.2.2. *Every component i of the potential function is $\phi_i \geq 0$, therefore $\phi \geq 0$.*

Proof. Since the potential function only sums terms for which $x_i < x_i^{DYN}$ we have that

$$\begin{aligned}\phi_i &= c_i x_i^{DYN} \ln \frac{(1+\delta)x_i^{DYN}}{x_i + \delta x_i^{DYN}} \\ &= c_i x_i^{DYN} \ln \frac{1+\delta}{x_i/x_i^{DYN} + \delta} \\ &> c_i x_i^{DYN} \geq 0\end{aligned}$$

□

Lemma 4.2.3. *The rate of decrease of the potential function is $\frac{d\phi}{dt} \leq -\frac{1}{2}$.*

Due to the technical nature of the proof, it is deferred to the paper [Anand et al., 2022a].

Next, we prove the bound that connects the potential function with the $DYNAMIC$ solution.

Lemma 4.2.4. *For every component of the potential function we have $\phi_i \leq c_i x_i^{DYN} \cdot O(\log k)$.*

Proof.

$$\begin{aligned}\phi_i &= c_i x_i^{DYN} \ln \frac{(1+\delta)x_i^{DYN}}{x_i + \delta x_i^{DYN}} \\ &= c_i x_i^{DYN} \ln (1/\delta + 1) \frac{x_i^{DYN}}{x_i/\delta + x_i^{DYN}} \\ &< c_i x_i^{DYN} \ln (1/\delta + 1) \\ &= c_i x_i^{DYN} \cdot O(\log k)\end{aligned}$$

□

At this point we are ready to prove the theorem concerning the consistency of the algorithm.

Theorem 4.2.1. *The cost incurred by the algorithm is* $ALG \leq O(\log k) DYNAMIC$.

Proof. The potential function decreases at 3 times the rate of increase of the cost incurred by the algorithm. The cost is initially equal to 0, while the potential function at worst reaches 0 starting from a positive value. Therefore

$$ALG = \sum_i c_i x_i \leq 3\phi(j = 0)$$

Due to the previous lemma we get

$$ALG \leq 3 \cdot O(\log k) \cdot \sum_i c_i x_i^{DYN} = O(\log k) \cdot DYNAMIC$$

Doubling the variables in order to obtain a feasible solution only doubles the total cost. □

Surprisingly, in order to obtain a robust algorithm, we only need to apply the framework itself to combine the solution described above, with that suggested by any online algorithm.

Theorem 4.2.2. *There is an algorithm whose cost is bounded by* $ALG \leq \min\{O(\log k) \cdot DYNAMIC, O(\log d) OPT\}$, *where* d *is the maximum number of variables with non zero coefficients in each constraint.*

Proof. Suppose that we use the framework in order to combine two solutions. One is the solution given by the online algorithm from [Gupta and Nagarajan, 2012] in each step, which is an $O(\log d)$ approximation. The other is the solution given by the framework when it is ran using the k predictions as suggestions in each step. As a total, the algorithm incurs a ratio of $O(\log 2) = O(1)$ over the $DYNAMIC'$, which is allowed to only choose from the two suggestions. But as discussed earlier any dynamic solution is upper bounded by its corresponding static. The two static solutions have bounds $O(\log d) OPT$ and $O(\log k) DYNAMIC$ which gives the theorem. □

Online Covering with Box Constraints. While the above analysis suffices for problems like weighted set cover and weighted caching, it lacks the ability to bound variables by each other, as in $x_{ij} \leq y_i$. Motivated by fractional facility location, where a set of variables y_i indicate the extent to which each facility is open and another set of variables x_{ij} which indicate how much the facility i covers demand j, we need extra constraints to express the fact that in order for a facility to cover a demand by x_{ij} it needs to be open by at least that much. The covering problem only allows for positive coefficients a_{ij}, therefore this case is not

covered by it and a new problem is needed, than the previous one. In its general formulation the covering problem with box constraints is

$$\begin{aligned} \min & \sum_i c_i y_i + \sum_i \sum_j d_{ij} x_{ij} \\ s.t. & \sum_i a_{ij} x_{ij} \geq 1 \quad \forall j \\ & x_{ij} \leq y_i \quad \forall i, j \\ & y_i \in [0,1] \quad \forall i \end{aligned}$$

In this case the online input is the constraints $sum_i a_{ij} x_{ij} = 1$, since everything else is known offline. The update of the x_{ij} to that of the x_i in the simple covering problem. The main difference is that this increase is only made when the corresponding box constraint $x_{ij} \leq y_i$ is strictly satisfied. Otherwise, both variables are increased at the same rate, which also incorporates the cost c_i of y_i.

Algorithm 5: Online Covering Framework with Box Constraints

```
forall steps j do
    Set Γ_ij = Σ_s x_ij(s) while Σ_i a_ij x_ij < 1/2 do
        forall i ∈ [n] such that x_ij < 1/2 do
            if x_ij < y_i then
                Increase x_i j with rate dx_ij/dt = (a_ij/d_ij) · (x_ij + δ · Γ_ij)
            else
                Increase both x_ij and y_i at rates ∂y_i/∂t = ∂x_ij/∂t = dx_ij/dt = (a_ij/(d_ij + c_i)) · (x_ij + δ · Γ_ij)
```

The analysis is very similar so we present only the theorem, which is obtained by combining an online algorithm with the framework, as before.

Theorem 4.2.3. *There is an algorithm for the online covering problem with box constraints that incurs cost* $ALG \leq \min\{O(\log k) DYNAMIC, \alpha OPT\}$*, where* α *is the approximation ratio of a known online algorithm for the specific problem.*

4.3 Online Graph Algorithms with Predictions

In their paper [Azar et al., 2021], Azar, Panigrahi, Touitou, design a framework with metric problems in mind. In their work, they attempt to create a black box method for combining algorithms, one for the online input and another for augmenting the solution with elements bought according to the predictions. The key to this framework is a new notion of error that they introduced, capturing both the spacial distance between two distributions of requests and the fact that some demands can be far enough from each other to be considered outliers.

Setting. As mentioned, we are concerned with problems defined on a metric space (M, d). In these problems we are given a set of requests R as input, and a universe S with the elements that a solution can buy in order to cover the requests. For example, in Facility Location the requests are the demands and the elements of the universe S are the locations where facilities can be opened. We note that problems on graphs can be described by this setting, by using the usual shortest path distance as the metric. This extends the framework to problems such as Steiner Tree/Forest. Since we are concerned with online problems, the requests R are assumed to be given one by one in an online fashion, as it is the case in Online Facility Location and Online Steiner Tree.

Predictions. This framework operates on predictions of the whole input. The algorithms designed are expected to be given a set $\hat{R}$ of predictions for all the requests as offline input. Then, using these predictions they should obtain offline solutions on them that they buy hoping that they will prove useful in covering future input.

Error Metric. In graph problems like Steiner Tree, the error metric of outliers alone has already been used in [Xu and Moseley, 2021]. In that setting, any predicted terminal that does not coincide with an actual terminal is considered flat-out wrong and contributes 1 to the error. Yet the general class of metric problems can also utilize the notion of distance between two distributions, by means of a matching. Motivated by these two facts, they combined these two notions to introduce the *metric error with outliers*. Specifically, the error is now a tuple (Δ, D), where Δ is the number of outliers, both predicted and actual input, and D the distance between requests that have been matched across the two distributions. If the matched subsets of $R, \hat{R}$ are $T, \hat{T}$, then $\Delta = |R \setminus T \cup \hat{R} \setminus \hat{T}|$. In essence, for every number of outliers Δ there are many different matchings that can be made and vice versa. These possible tuples create a Pareto frontier. For instance, for a fixed Δ we can acquire the least D by using some variation of min cost matching that fits the given problem. Then, the algorithms are analyzed against any possible value that these two values can have. The consistency of the algorithms is obtained by setting $\Delta = 0$ and $D = \epsilon$, while the robustness is obtained for $\Delta = |R \cup \hat{R}|$.

Requirements. If we wish to utilise the offline predictions of the input we will need to run some offline algorithm on them, while another online algorithm handles the input. In accordance with the outliers in the error metric, we need to be able to discard predictions. In fact, it would be preferrable to be able to tune how many of the predicted requests we wish to cover, while keeping the cost incurred as low as possible. This is where prize-collecting algorithms prove to be useful. If we were to assign the same penalty π to all predicted requests and then run an approximation algorithm, then by increasing π we would be getting more and more requests covered, as the penalty becomes more inhibiting. By repeatedly running this algorithm we could find the point where we get the approximately lowest possible number of predictions that can be covered by a solution that spends no more than the price of the online solution. Ideally, the offline algorithm would recognise which predictions are bad and would only consider the rest of them. While that is not possible, this framework successfully buys solutions that incur no more cost than it would if it knew which are to be matched. As for the online algorithm, it will similarly not be aware of the matching and which solutions could cover which parts of the online input. Nevertheless, if it has the property of being subset competitive we could analyze it with respect to the matched requests it correctly recognised. Therefore, the two required algorithms for this framework are the following

- A constant γ-approximation offline algorithm for the prize-collecting variation of the problem.
- A subset competitive online algorithm for the problem with approximation ratio $f(|R|)$.

Algorithm. As mentioned before, we will need to combine offline solutions bought for the predicted instance with online solutions bought for the input. In order to achieve that, we can request an offline solution every time the cost incurred by the online algorithm has doubled. This solution is expected to cost no more than that of the online solution, while covering as many predicted requests as possible.

The subroutine $PARTIAL$ is responsible for buying elements that cover the predicted requests, leaving approximately u of them not satisfied. By exploring different penalties, it finds those that leave close to u requests. By running it multiple times for different values of u we maximize the satisfied predictions while also incurring cost within a constant ratio from the online algorithm.

Algorithm 6: General Framework for Online Algorithms with Predictions

Input: ON – An online algorithm for the problem
PC – a γ-approximation for the prize-collecting problem
$\hat{R}$ – a prediction of requests

Initialization
  Initialize $B \leftarrow 0$, $\hat{B} \leftarrow 0$ and $S \leftarrow \emptyset$.
  Initialize ON to be a new instance of the online algorithm.

forall *requests r in the input* **do**
  Send r to the online algorithm ON, and augment S accordingly. Increase B by the resulting cost incurred by ON.
  // Whenever the cost of the online algorithm doubles, buy another offline solution
  if $B \geq 2\hat{B}$ **then**
    Set $\hat{B} \leftarrow B$
    Calculate $0 \leq u \leq |\hat{R}|$, the minimum number such that $c(\textsc{Partial}(\hat{R}, u)) \leq 3\gamma\hat{B}$.
    Augment S with the elements of $\textsc{Partial}(\hat{R}, u)$.
    Start a new instance of the online algorithm ON where $c(e) = 0$ for all $e \in S$.

Function $\textsc{Partial}(\hat{R},u)$
  if $\gamma u \geq |\hat{R}|$ **then return** $\emptyset$
  For every x, define π_x to be the prize-collecting penalty that penalizes every request by x.
  Let i be the minimum integer such that $PC(\hat{R}, \pi_{2^i})$ does not satisfy $\leq \gamma u$ requests.
  Let $S_1 = PC(\hat{R}, \pi_{2^{i-1}})$ and $S_2 = PC(\hat{R}, \pi_{2^i})$.
  Let u_1, u_2 be the number of requests from $\hat{R}$ which are not satisfied by S_1, S_2, respectively.
  if $\gamma u \geq \frac{u_1+u_2}{2}$ **then return** S_1 **else return** S_2

Analysis of $PARTIAL$. First, we analyze the $PARTIAL$ subroutine in order to prove that it achieves what it set out to.

Lemma 4.3.1. *Running $PARTIAL$ with input u gives the following*

- *The predicted requests not satisfied by the solution are at most $2\gamma u$.*
- *The solution costs at most $3\gamma c(S^*)$ where $c(S^*)$ is the cost of the optimal solution that doesn't cover u predicted requests.*

Proof. First of all, notice that the subroutine chooses S_1 or S_2 based on whether γu is closer to u_1 or u_2 respectively. On one hand, $u_2 \leq \gamma u \leq 2\gamma u$. On the other, when S_1 is chosen, γu is closer to u_1, therefore u_1 cannot exceed $2\gamma u$.

As for the bound of the cost, we use the approximation ratio of algorithm PC to get $c(PC(\pi_x)) \leq \gamma c(S^*_{\pi_x})$ where $c(S^*_{\pi_x})$ is the optimal solution of the prize collecting with penalties π_x. Note that S^* is the optimal solution leaving $u^* \leq u$ requests not satisfied with given penalties. Given that it is a valid solution the bound becomes $c(PC(\pi_x)) \leq \gamma(c(S^*) + x \cdot u^*)$.

Applying it to the two solutions S_1 and S_2 we get

$$c(S_1) + 2^{i-1} \cdot u_1 \leq \gamma c(S^*) + 2^{i-1} \cdot \gamma u^* \tag{4.1}$$

$$c(S_2) + 2^i \cdot u_2 \quad \leq \gamma c(S^*) + 2^i \cdot \gamma u^* \tag{4.2}$$

When $S = S_1$ the lemma follows because $\gamma u \leq u_1$ and

$$c(S_1) \leq \gamma c(S^*) + 2^{i-1} \cdot (\gamma u^* - u_1) \leq \gamma c(S^*) + 2^{i-1} \cdot (\gamma u - u_1) \leq \gamma c(S^*)$$

When $S = S_2$ we cannot directly compare γu^* with u_2 as we did with u_1. In order to acquire a bound we introduce γu in inequality (2)

$$c(S_2) \leq \gamma c(S^*) + 2^i(\gamma u^* - u_2) = \gamma c(S^*) + 2^i(\gamma u^* - \gamma u) + 2^i(\gamma u - u_2) \leq \gamma c(S^*) + 2^i(\gamma u - u_2)$$

Using the fact that $\gamma u \leq \frac{u_1+u_2}{2}$

$$c(S_2) \leq \gamma c(S^*) + 2^i \frac{u_1 - u_2}{2}$$

In order to bound $u_1 - u_2$ we use inequality (1)

$$\gamma c(S^*) \geq c(S_1) + 2^{i-1}(u_1 - \gamma u) \geq 2^{i-1} \frac{u_1 - u_2}{2}$$

Using the last two inequalities we get the lemma. □

Analysis of the framework. Having analyzed $PARTIAL$ and the quality of the solutions it obtains, it is time to discuss how these are combined with the online solutions to give some general bounds.

We notice that in the beginning, the algorithm is expected to buy enough solutions for the future relying heavily on the predictions, since the online algorithm does not yet have enough information to make good decisions. After a certain point, though, when enough of the input has been observed and enough offline solutions have been bought, the online algorithm will have to decide whether to use them or not. This will be decided for the most part by the error of the predictions. If the predictions were good then all the algorithm does is to connect them to the requests. On the opposite case, the online algorithm will have to buy new ones. Motivated by this, we split the execution in two parts, suffic and prefix, before and after a major iteration i. This iteration will have to be defined by every application of the framework and should be chosen so that the total cost incurred up to that point is close to OPT, so that the offline solutions will have covered most of the predictions.

In order to refer to the cost up to a certain iteration, we use B_j and $\hat{B}_j$ for the value of B and $\hat{B}$ respectively, after iteration j. For the purposes of the analysis, we can split the execution of the online algorithm after iteration i in different phases $ON_0, \ldots, ON_m$, each one ending on an iteration when the algorithm has doubled its total cost and seeks offline solutions. These iterations are called *major* iterations and are denoted $i_j^\star$.

Lemma 4.3.2. *The cost of the iterations up to i is $Prefix(i) \leq O(1) \cdot OPT + (12\gamma + 2) \cdot \hat{B}_{i-1}$*

Proof. The cost of the iterations up to i is made up of the cost incurred by the online algorithm, which is B_i and that of the offline solutions, which is bounded by $3\gamma\hat{B}_j$ if j was a major iteration. Adding all these costs together we get

$$Prefix(i) \leq B_i + 3\gamma \sum_{j=0}^{i} \hat{B}_j = B_i + 3\gamma\hat{B}_i + 3\gamma \sum_{j=0}^{i-1} \hat{B}_j$$

The reason why we separate costs corresponding to iteration i from the rest, is because i is a major iteration and $\hat{B}_i = B_i = B_{i-1} + ON(r_i)$. But using subset competitiveness of the online algorithm it is clear that $ON(r_i) \leq f(1) \cdot OPT = O(1) \cdot OPT$. On iterations that are not major, like $i - 1$ the invariant $B_{i-1} < 2\hat{B}_{i-1}$ holds. Using all these we get

$$B_i + 3\gamma\hat{B}_i = (1+3\gamma)B_i = (1+3\gamma)\cdot(B_{i-1} + O(1)\cdot OPT) \leq (2+6\gamma)\cdot\hat{B}_{i-1} + O(1)\cdot OPT$$

As for the sum, due to the exponential increase of $\hat{B}$ we can bound $\hat{B}_j \leq (\frac{1}{2})^{i-1-j}\hat{B}_{i-1}$ for $j \leq i-1$. In order to sum all these we notice the geometric series and we get

$$3\gamma\sum_{j=0}^{i-1}\hat{B}_j \leq 6\gamma\hat{B}_{i-1}$$

which proves the lemma. □

Lemma 4.3.3. *The cost of the iterations starting with* $i{+}1$ *is* $Suffix(i) \leq O(1)\cdot\max\{ON_{m-1}, ON_m\}$

Proof. The cost after iteration i is expected to be bound by that of the online in its last phases, since the cost required to conclude a phase is equal to that incurred from the beginning until the last phase, i.e. $\hat{B}_{i^\star_{j+1}} - \hat{B}_{i^\star_j} \geq 2\cdot\hat{B}_{i^\star_j} - \hat{B}_{i^\star_j} = \hat{B}_{i^\star_j}$

$$Suffix(i) = \sum_{j=0}^{m} ON_j + 3\gamma\sum_j \hat{B}_{i^\star_j}$$

The cost of the offline solutions can be bounded as follows

$$3\gamma\sum_j \hat{B}_{i^\star_j} \leq 3\gamma\sum_j\left(\hat{B}_{i^\star_{j+1}} - \hat{B}_{i^\star_j}\right) = 3\gamma\sum_{j=0}^{m} ON_j$$

What remains is to bound the cost of the online algorithm in its various phases. The argument is similar since

$$ON_j = B_{i^\star_{j+1}} - B_{i^\star_j} = \hat{B}_{i^\star_{j+1}} - \hat{B}_{i^\star_j} \geq \hat{B}_{i^\star_j}$$

For $j = m-1$ we have

$$ON_{m-1} \geq \hat{B}_{i^\star_{m-1}} \geq \sum_{j=0}^{j=m-2} ON_j$$

Therefore the sum up to $j = m-1$ is at most twice the cost of ON_{m-1}.

The next phase, ON_m is of unknown cost, since the input could end after one request, or even get to the point where the cost has doubled. Therefore, we use the highest of ON_{m-1}, ON_m as the bound, which gives the lemma. □

Having proved the main properties of the framework, it can be seen that in order to use it and obtain bounds for a given problem, one needs to choose the iteration i and give two bounds, one for $\hat{B}_{i-1}$ and another for ON_j. Of course, the definition of a proper error for the matching is a more intricate problem, one that we explore in the final section.

Chapter 5

Learning Augmented Algorithms

In this section we present a survey of some main results in the area of learning augmented algorithms, using both problem specific methods as well as the frameworks presented in the previous section.

5.1 Ski rental

The *Ski Rental* problem is simple enough to provide a warm up for the study of learning augmented algorithms, while at the same time requiring non trivial approaches to achieve good results. As a reminder, the setting is that every day we decide whether to rent the skis for a cost of 1, or to buy them once and for all for a cost of b. The problem is that we do not know how many days we will be skiing, which is exactly why we choose to predict it. A simple approach that yields a 2 approximation is to rent until day $b-1$ and then buy. There is also a randomized algorithm by [Karlin et al., 1990] for the problem achieving ratio 1.58. Both of these algorithms are optimal.

5.1.1 Single Prediction

Considering that we have a prediction $\hat{x}$ of the total days x, we need to find a proper way to leverage it with respect to the error $\eta = |\hat{x} - x|$ in order to achieve better than the previous two deterministic and randomized results in the worst case. The algorithms of this section were given in [Purohit et al., 2018].

Deterministic algorithm. If we were to assume that the prediction is trustworthy, we could implement the following simple rule. If $\hat{x} \geq b$ then pay b to buy the skis, otherwise rent until the end of the input, paying x.

Lemma 5.1.1. *If the above algorithm incurs cost ALG, then* $ALG \leq OPT + \eta$.

Proof. A simple case study exposes the weaknesses of this approach.

- If $\hat{x}$, x are both less or both greater than b, then $ALG = OPT$
- If $x < b \leq \hat{x}$ then $OPT = x$, $ALG = b \leq \hat{x} \leq \hat{x} - x + x = OPT + \eta$
- If $\hat{x} < b \leq x$ then $OPT = b$, $ALG = x = x - b + b \leq b + x - \hat{x} = OPT + \eta$

□

If the prediction is actually good, then the above algorithm achieves OPT. In the opposite case, its ratio is unbounded. As an example, given $\hat{x} < b$ and $x = k \cdot b \gg b$ the ratio is k because the algorithm blindly trusts that the total days are less than b and keeps renting. Similarly, given $\hat{x} > b$ and $x = 1$ the algorithm buys immediately, giving ratio b. Both these

issues could be alleviated by imposing a limit on how long the algorithm can keep renting and how early it can start buying. Introducing a tunable parameter $\lambda \in (0,1)$ we can express our trust in the predictions. Specifically, if we were to buy, i.e. $\hat{x} \geq b$ we rent until λb and then buy. On the other hand, if we were to rent, i.e. $\hat{x} < b$, we only do so until $\frac{b}{\lambda}$ and then buy.

Theorem 5.1.1. *The modified algorithm incurs cost* $ALG \leq \min\{(1+\frac{1}{\lambda}) \cdot OPT, (1+\lambda) \cdot OPT + \frac{\eta}{1-\lambda}\}$

Proof. We proceed with a similar case study, in order to prove both bounds.

- If $\hat{x} \geq b$
  - If $x < \lceil \lambda b \rceil$ then $ALG = OPT$ because the algorithm does not buy before λb
  - If $x \geq \lceil \lambda b \rceil$ then $ALG \leq (1+\lambda)b$, while OPT is either x or b depending on whether x exceeds b. The worst ratio corresponds to $x = \lceil \lambda b \rceil$ and is $1 + \frac{1}{\lambda}$. On the other hand, it is also true that $b \leq OPT + \eta$, therefore $ALG \leq (\lambda+1)(OPT+\eta)$
- If $\hat{x} < b$
  - If $x \leq b$ then $ALG = OPT$ because the algorithm keeps renting until day x
  - If $b < x \leq \lceil \frac{b}{\lambda} \rceil$ then $OPT = b$, while $ALG \leq \frac{1}{\lambda}$ and $ALG = x = x - b + b \leq OPT + \eta$
  - If $x > \lceil \frac{b}{\lambda} \rceil$ then $OPT = b$ while $ALG \leq b + \frac{b}{\lambda}$. Obviously, $ALG \leq (1+\frac{1}{\lambda}) \cdot OPT$. On the other hand, $\eta = x - \hat{x} > \frac{b}{\lambda} - b = (1-\lambda)\frac{b}{\lambda}$. As a result, $ALG \leq OPT + \frac{\eta}{1-\lambda}$

□

The above analysis, guarrantees $1 + \frac{1}{\lambda}$-robustness and $1+\lambda$-consistency. Depending on our trust in the predictions we decide which side of the tradeoff we will favour. High values for λ indicate low trust in the predictions but provide increased robustness.

Randomized algorithm. As for the randomized algorithm, the ideas that drive its design build upon the previous discussion, by selecting the day that we purchase by defining proper distributions. The limitations of λb and $\frac{b}{\lambda}$ are still present but we no longer wait until we reach these specific days, so that the adversary cannot force the algorithm's choices by giving a bad prediction. Specifically, if the prediction is $\hat{x} \geq b$ the algorithm a day before λb to buy, while if $\hat{x} < b$ the algorithm will choose a day before $\frac{b}{\lambda}$. Before giving the exact distributions we will attempt to provide some intuition into their construction.

Suppose that $\lambda = 1$ which means $\lambda b = \frac{b}{\lambda} = b$ matching the two limits. This is the original worst case scenario where no prediction is used. We need a distribution p_i that expresses how long we should wait before buying, for each day $i \in [1, b]$. This p_i calculates the probability that $x \leq i$, implying that if we rent until i we will not have to buy at all. Since we do not know x and assuming it is $x \leq b$, selecting it at random would give i with probability $\frac{1}{b}$ and anything but i with probability $1 - \frac{1}{b}$. Therefore, the probability that it is $x \in [1, i]$ is equal to the probability of $x \notin [i+1, b]$ which is to say $P(x \in [1,i] \mid x \in [1,b]) = (1-\frac{1}{b})^{b-i}$. In order to get rid of the condition we have $P(x \in [1,i]) = (1-\frac{1}{b})^{b-i} \cdot P(x \in [1,b])$. Notice that $P(x \in [1,b])$ is constant with respect to i, therefore by summing we have $P(x \in [1,b]) = \frac{1/b}{1-(1-1/b)^b}$ and

$$p_i = (1-1/b)^{b-i} \cdot \frac{1}{b(1-(1-1/b)^b)}$$

In order to introduce λ we have to scale the distribution so that it is defined on different domains, $i \in [1, \lambda b]$ and $i \in [1, b/\lambda]$. This is simple, since we can adjust the exponent with

$k = \lceil \lambda b \rceil$ and $l = \lceil b/\lambda \rceil$ and get a different corresponding constant in each case which will give

$$q_i = (1-1/b)^{k-i} \cdot \frac{1}{b(1-(1-1/b)^k)}, \ i \leq k$$
$$r_i = (1-1/b)^{l-i} \cdot \frac{1}{b(1-(1-1/b)^l)}, \ \ i \leq l$$

Theorem 5.1.2. *The randomized algorithm achieves* $\mathbb{E}[ALG] \leq \min\{\frac{1}{1-e^{-(\lambda-1/b)}} \cdot OPT, \frac{\lambda}{1-e^{-\lambda}} \cdot (OPT+\eta)\}$ *for* $\lambda \in [\frac{1}{b}, 1]$.

Proof. We proceed by a case study

- If $\hat{x} > b$ and $x \leq k$ then we will have to rent the skis for $i-1$ days and eventually buy them on the i-th day. As before we have $b \leq OPT + \eta$ and

$$\begin{aligned}\mathbb{E}[ALG] = \sum_{i=1}^{k} (b+i-1) \cdot q_i &= \frac{k}{1-(1-1/b)^k} \\ &\leq \frac{k}{1-e^{-k/b}} \\ &\leq \frac{k/b}{1-e^{-k/b}} \cdot (OPT+\eta) \\ &\leq \frac{\lambda}{1-e^{-\lambda}} \cdot (OPT+\eta)\end{aligned}$$

- If $\hat{x} > b$ but $x < k$ then for $i \leq x$ we pay $b+i-1$ as before but beyond that point we only pay x as we did not have to buy the skis. In this case $OPT = x$ and

$$\begin{aligned}\mathbb{E}[ALG] = \sum_{i=1}^{x} (b+i-1) \cdot q_i + \sum_{i=x+1}^{k} x \cdot q_i &= \frac{x}{1-(1-1/b)^k} \\ &\leq \frac{1}{1-e^{-k/b}} \cdot OPT \\ &= \frac{1}{1-e^{-\lambda}} \cdot OPT\end{aligned}$$

  The first bound is given because $\lambda \leq \frac{1}{b}$ therefore $\mathbb{E}[ALG] \leq \frac{1}{1-e^{-(\lambda-1/b)}} \cdot OPT$. The second bound is given because $OPT = x \leq \lambda b \leq \lambda \hat{x} = \lambda(OPT+\eta)$ which implies $\mathbb{E}[ALG] \leq \frac{\lambda}{1-e^{-\lambda}} \cdot (OPT+\eta)$

The proofs for $\hat{x} < b$ follow the exact same approach and give the same bounds. □

5.1.2 Multiple Predictions

Now we can consider a different scenario, where we receive k predictions $\hat{x}_i$ of x as presented in [Gollapudi and Panigrahi, 2019]. In this case, the error will be that of the best prediction $\eta = \min_i\{|\hat{x}_i - x|\}$. In order to understand the problem, we start by assuming that $\eta = 0$, or equivalently that one of the prediction is correct, in order to derive a consistent algorithm for the problem.

To begin with, suppose that $k = 2$ and the predictions are $\hat{x}_1 \leq \hat{x}_2$. Then one of them is equal to x. If both predictions are on the same side of b, then the algorithm can always make the optimal decision, as we will know where x is. On the other hand, if $x_1 < b \leq x_2$, then we need a rule based on which to decide whether to buy or rent and until when.

First approach. One natural approach would be to rent until $\hat{x}_1$, at which point we will know whether it is the correct prediction or not, and then buy if the input continues. Obviously, when it is the correct prediction $\hat{x}_1 = x$, we get the optimal solution which means ratio 1. Otherwise, $b \leq \hat{x}_2 = x$ and $OPT = b$ but we pay $ALG = x + b$. The ratio will be $\frac{\hat{x}_1+b}{b}$ in this case. The worst ratio will be achieved when $\hat{x}_1 = b - \epsilon$ in which case it will be $2 - \epsilon$, which does not improve over the online algorithm without predictions.

Consistent algorithm for $k = 2$. The previous algorithm despite its pitfalls, offers some intuition over the problem. Using $\hat{x}_1$ as the decision boundary for buying and renting, the ratios in the two different cases are unbalanced. By moving this boundary, we expect to suffer some ratio worse than one when $\hat{x}_1$ is the correct prediction but we will also improve in the case when $\hat{x}_2$ is correct, since we will not have to rent until $\hat{x}_1$. In essence we are looking for a x_1 that balances the two ratios, which is necessary, since the worst one counts. If $\hat{x}_1$ appears before it we will rent until x_1, otherwise we will buy.

Theorem 5.1.3. *This algorithm achieves consistency equal to $\phi = \frac{\sqrt{5}+1}{2} = 1.618\ldots$*

Proof. If $\hat{x}_1 = x$, then the worst ratio is obtained when $\hat{x}_1 > x_1$ since we will buy, resulting in a ratio of $\frac{b}{\hat{x}_1} < \frac{b}{x_1}$. Otherwise if $\hat{x}_2 = x$, then the worst ratio will be obtained if $\hat{x}_1 < x_1$ since we will have to rent and then buy, resulting in a ratio of $\frac{b+x_1}{b}$. We can balance these two by solving the equation $\frac{b+x_1}{b} = \frac{b}{x_1}$ with respect to x_1 which gives $x_1 = \frac{\sqrt{5}-1}{2} \cdot b$ and a consistency of ϕ. □

We remark that this is the best possible for $k = 2$.

Consistent algorithm for any k. What remains is to generalise the above algorithm for any number of predictions. We continue to assume that one prediction is correct. In this case we will have separate the segment $[1, b]$ in $k-1$ intervals using x_i so that all the ratios will be balanced. As for the algorithm, when $k = 2$ we would only start renting if a prediction was present inside the first interval $[1, x_1)$ and no prediction was in the second interval $[x_1, b)$. In the opposite case we would buy. Therefore, a natural generalisation is to inspect these intervals from left to right, renting until the first interval without a prediction in it and then buying. In essence, regardless of the predictions we will have split the segment $[1, b)$ into intervals independent of the predictions and once we are given the predictions we will choose which of those intervals will be the boundary between renting and buying.

Theorem 5.1.4. *The above algorithm achieves a consistency that results from the equation $c_k = \sum_{i=1}^{k-1} c_k^{-i}$*

Proof. If there is no prediction in the first interval the algorithm will buy immediately, while the optimal solution in the worst case will buy at x_1, the next boundary, giving a ratio of $\frac{b}{x_1}$. If we find no prediction in the i-th interval, which is $[x_i, x_{i+1})$ then $ALG = x_i + b$ while $OPT \geq x_{i+1}$ giving a ratio no worse than $\frac{x_i+b}{x_{i+1}}$. In order to balance them we solve

$$\frac{b}{x_1} = \frac{b + x_1}{x_2} = \cdots = \frac{b + x_{k-1}}{b}$$

The solution of these equations is

$$x_j = b \cdot \sum_{i=1}^{j} \left(\frac{x_1}{b}\right)^i$$

where x_1 is calculated by

$$1 = \sum_{i=1}^{k} \left(\frac{x_1}{b}\right)^i$$

Now suppose that the first interval without a prediction in it is $[x_i, x_{i+1})$. If $x \leq x_i$ then the algorithm achieves the optimal cost as it rents until x_i. Otherwise $x \geq x_{i+1}$. If $x < b$ then the ratio is $\frac{x_i+b}{x} \leq \frac{x_i+b}{x_{i+1}}$, otherwise it is $\frac{x_1+b}{b} \leq \frac{x_i+b}{x_{i+1}}$. So in both cases, the ratio is equal to $c_k = \frac{b}{x_1}$ which can be calculated by the last equation as

$$c_k = \sum_{i=1}^{k-1} c_k^{-i}$$

□

As before this is the best possible consistency for k predictions.

Consistent and robust algorithm for any k. The last step will be to allow for error. Once again, we will attempt to split $[1, b)$ into intervals and rent until the first one without prediction. As in the case of a single prediction, the worst scenario is that no prediction is found in the first interval, which will force us to buy, while x could be inside that interval, leading to a ratio $\frac{b}{x}$. If $x = 1$ this ratio is exactly b. To resolve this, we impose a tunable limit λb, before which we always rent. This could be considered the first boundary.

Theorem 5.1.5. *The algorithm above incurs cost $ALG \leq \min\{1 + \frac{1}{\lambda}, \bar{c}_k\} \cdot OPT$ where $\bar{c}_k = \sum_{i=1}^{k-1} (\bar{c}_k)^{-i} + \lambda \cdot \bar{c}_k^{-k}$.*

Proof. As before the analysis balances the ratios, with the difference that now the first interval is $[1, \lambda b)$, in which case

$$\frac{b + \lambda b}{x_1} = \frac{b + x_1}{x_2} = \cdots = \frac{b + x_{k-1}}{b}$$

By solving these equations we get the following consistency ratio

$$\bar{c}_k = \sum_{i=1}^{k-1} (\bar{c}_k)^{-i} + \lambda \cdot \bar{c}_k^{-k}$$

As for the robustness, the idea is the same as in the single prediction. We will not buy before λb, therefore the worst ratio is obtained when x comes right after we buy, giving a ratio of $\frac{\lambda b+b}{\lambda b} = 1 + \frac{1}{\lambda}$.

□

5.2 Job Scheduling

In this section we will consider the problem of job scheduling as studied in [Purohit et al., 2018]. Having n jobs with required execution times denoted by $x_1, x_2, \ldots, x_n$, we wish to execute them on a single machine and need to determine their order. The objective with respect to which we need to optimize is the sum of completion times for all jobs.

In the *clairvoyant* case, we are given these execution times and it can easily be seen that the optimal strategy is to schedule them in a non decreasing order of their execution times (*SJF*). Intuitively, any long execution time spent in the beginning will be charged multiple times, therefore swapping it with a lower execution time will only reduce the total cost. The scenario that will be studied, will be the *non clairvoyant* one where we do not know any execution times until the execution of a specific job has finished. In this case it is necessary to be able to preempt and resume jobs, which is a reasonable requirement since we have absolutely no information about the jobs and could potentially end up getting stuck in the

execution of very long jobs early on. The optimal approximation in this setting is *Round Robin*, which was proved to be a 2-approximation in [Motwani et al., 1994]. This algorithm runs the k jobs remaining at the same rate $\frac{1}{k}$, meaning that for infinitessimal time windows, every job is ran for a $\frac{1}{k}$ fraction of the available time and then swapped by the next.

5.2.1 Preferential Round Robin Algorithm

In the analysis below, these two strategies will be combined in order to utilise predictions of the execution times and achieve ratios better than 2. A useful assumption w.l.o.g. is that all execution times are no less than 1, which can be achieved with normalisation, if there are no zero-length jobs.

Predictions. The predictions will give the execution times of each job, denoted by $\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n$.

Error metric. The error will be the natural L_1-norm between the predictions and the actual execution times $\eta = \sum_i \eta_i = \sum_i |x_i - \hat{x}_i|$.

Starting with the consistency, if the predictions were perfect, then the algorithm we would use would be *SJF* on the predictions, which we will call *SPJ* and we would obtain the optimal solution. On the other hand, when the predictions are bad we would use the best approximation available which is *Round Robin* with ratio equal to 2. Given that we can run the jobs at different rates, we can run both of these algorithms adjusting the rates they would assign to the different jobs by a parameter $\lambda \in [0, 1]$.

At this point it would be useful to define *monotonicity*, a property that allows us to combine two arbitrary algorithms in this way.

Definition 5.2.1. *An algorithm is* monotone *if given instances $x_1, x_2, \ldots, x_n$ and $x'_1, x'_2, \ldots, x'_n$ with $x_i \leq x'_i$ for all i, it would not incur any more cost in the first instance than the second one.*

Both *Round Robin* and *SPJF* are monotonic, since they both make the same choices regardless of input, for fixed predictions, therefore the total cost could only decrease due to jobs finishing earlier.

Now we give the lemma for the combination of any two *monotonic* algorithms.

Lemma 5.2.1. *If we adjust the rates assigned to jobs by a* monotonic *α-approximation algorithm by λ and those by a* monotonic *β-approximation algorithm by $1 - \lambda$ then the new ratio would be* $\min\{\frac{\alpha}{\lambda}, \frac{\beta}{1-\lambda}\}$.

Proof. Assume that only the first algorithm ran. Adjusting the rates of its jobs by a factor λ means that we multiply their rates, slowing them down. In terms of completion times, they are delayed by a factor of $\frac{1}{\lambda}$ and the new approximation ratio would be $\frac{\alpha}{\lambda}$. The fact that we are also running the second algorithm does not increase this ratio, as it can only decrease the execution times that the first algorithm sees and due to monotonicity this would only decrease the ratio. Using the same logic on the second algorithm and multiplying its rates by $1 - \lambda$ we get the lemma. □

In essence, λ indicates our trust in each algorithm. A high value of λ trusts the α-approximation as it slows down the rates of the β-approximation.

At this point, it remains to prove the approximation ratio of *SPJF*.

Lemma 5.2.2. *The* SPJF *algorithm achieves an approximation ratio of* $1 + \frac{2\eta}{n}$.

Proof. Suppose that the actual execution times are indexed in a non decreasing order $x_1 \leq x_2 \leq \cdots \leq x_n$.

Any algorithm will have to pay the sum of the execution times $\sum_i x_i$ in order to actually process the jobs. In order to measure completion times, we need to add the time it took for jobs executed before it. In order to measure that, we define $d(i,j)$ which is how much of i was executed before j finished, meaning how much job i delayed job j.

As for the total cost, each job i is delayed by each j that would come before it in the optimal solution by $d(i,j)$, which can be directly charged to it. But also delays those j that would be executed before it in the optimal solution, by $d(j,i)$ which will be bounded by the error.

The algorithm will prioritize the jobs with short predicted execution times. For $i < j$ and $\hat{x}_i < \hat{x}_j$, then j does not delay i, while i delays j by its actual execution time x_i. On the other hand, if $\hat{x}_i \geq \hat{x}_j$, then only job j delays job i by x_j.

$$\begin{aligned} ALG &= \sum_{j=1}^{n} \left[x_j + \sum_{i<j} d(i,j) + d(j,i) \right] \\ &= \sum_{j=1}^{n} x_j + \sum_{\{i<j|\hat{x}_i<\hat{x}_j\}} x_i + \sum_{\{i<j|\hat{x}_i\geq\hat{x}_j\}} x_j \\ &= \sum_{j=1}^{n} x_j + \sum_{i<j} x_i + \sum_{\{i<j|\hat{x}_i\geq\hat{x}_j\}} (x_j - x_i) \end{aligned}$$

Now observe that $\sum_{j=1}^{n} x_j + \sum_{i<j} x_i$ is exactly the cost of the optimal solution. Therefore

$$\begin{aligned} ALG &= OPT + \sum_{\{i<j|\hat{x}_i\geq\hat{x}_j\}} (x_j - x_i) \\ &= OPT + \sum_{\{i<j|\hat{x}_i\geq\hat{x}_j\}} (x_j - \hat{x}_j + \hat{x}_i - x_i + \hat{x}_j - \hat{x}_i) \\ &\leq OPT + \sum_{\{i<j|\hat{x}_i\geq\hat{x}_j\}} (x_j - \hat{x}_j + \hat{x}_i - x_i) \\ &= OPT + \sum_{\{i<j|\hat{x}_i\geq\hat{x}_j\}} (\eta_j - \eta_i) = OPT + (n-1)\eta \end{aligned}$$

Then the ratio is $1 + \frac{(n-1)\eta}{OPT}$ but all execution times are no less than 1 therefore $OPT \geq \frac{n(n+1)}{2}$ and the ratio in the worst case is $1 + \frac{2(n-1)\eta}{n(n+1)} < 1 + \frac{2\eta}{n}$. □

Theorem 5.2.1. *The algorithm that combines the* SPJF *and* Round Robin *has a ratio* $\min\{\frac{1}{\lambda}(1 + \frac{2\eta}{n}), \frac{2}{1-\lambda}\}$.

5.3 Online Steiner Tree

Steiner Tree is the classical network problem, where we need to connect a subset of the nodes of a graph with edges of the minimum total cost. Formally, given a graph $G(V,E)$ and a subset $R \subseteq V$ of terminals of size $k = |R|$, the goal is to connect all terminals with edges of minimum total cost. In its online version, the terminals are given one by one, in the order $r_1, r_2, \ldots, r_k$ and on every step we need to connect the latest terminal to the rest by irrevocably buying edges. We refer to the set of terminals given up to step i as R_i. Since proved to be NP-Hard [Karp, 1972], a lot of effort has been put into designing approximation

algorithms both online and offline. One common technique used in order to approach it as a metric problem, is to consider the metric closure of the graph, effectively turning it into a complete graph whose edge costs are the shortest path distances between the nodes.

Lemma 5.3.1. *The offline algorithm [Kou et al., 1981] achieves a* 2*-approximation by finding the MST among the terminals.*

Lemma 5.3.2. *The greedy online algorithm [Imase and Waxman, 1991] which connects every new terminal* r_i *to* R_{i-1} *by buying the shortest path to a node in* R_{i-1} *is* $O(\log k)$ *competitive.*

The prediction used in both cases will be the input, i.e. the terminals. That is to say a set $\hat{R} \subseteq V$ is given as offline input.

5.3.1 Using Outliers as the Error

In the work of B. Moseley [Xu and Moseley, 2021], the Steiner Tree problem was studied for the first time in this setting. In this work, the error metric was chosen to be solely the number of mispredicted terminals.

Error Metric. The error is defined as $\eta = k - |R \cap \hat{R}|$.

η-Competitive Algorithm. As a natural first approach, we combine the two algorithms for the online and the offline case. Specifically, if a terminal in the input was not in the predictions we connect it with the shortest path incurring logarithmic cost on the error. Otherwise we buy a path from the MST on $\hat{R}$ that connects the terminal to another predicted terminal that appeared earlier in the input. To simplify the analysis, we assume that the predicted instance is of equal size k to the input, which can be lifted, as shown in the paper.

Algorithm 7: Online Algorithm with Predicted Terminals

```
1 forall r_i in the input do
2     if r_i ∉ R̂ or r_i is the first predicted terminal to appear as input then
3         Buy the shortest path that connects r_i with R_{i-1}.
4     else if r_i ∈ R̂ then
5         Buy a path from the MST(R̂) which connects r_i with a terminal in R_{i-1} ∩ R̂
```

The solutions bought for terminals that were not predicted make up the set A_1, while the rest make up the set A_2. These two sets obviously partition the solutions bought by the algorithm.

Lemma 5.3.3. *The cost incurred by line 3 of the algorithm is* $c(A_1) \leq \log \eta \cdot OPT$.

Proof. This follows directly from the greedy online algorithm. The terminals that satisfy the condition of line 3 are equal to the number of mispredicted terminals plus one for the first predicted terminal that appears in the input, which is $\eta + 1$ terminals. If the optimal solution on them is OPT' then $OPT' \leq OPT$ and due to the analysis of the greedy algorithm we get

$$c(A_1) \leq O(\log \eta) \cdot OPT' \leq O(\log \eta) \cdot OPT$$

□

Lemma 5.3.4. *Every edge* $e \in A_2$ *incurs no more cost than* OPT.

Proof. The edges of A_2 are bought due to terminals $r_i \in R_{i-1} \cap \hat{R}$, as paths of $MST(\hat{R})$. For every other terminal $r_j \in R_{i-1} \cap \hat{R}$ we have that $d(r_i, r_j) \leq OPT$, since these two terminals are connected by a path of G.

The path P_i chosen in line 5 is either the edge $e = (r_i, r_j)$, if it is included in the $MST(\hat{R})$, or $e \notin P_i$ because it was not included in the $MST(\hat{R})$. In the second case, adding e to $MST(\hat{R})$ creates a cycle. If this cycle included an even more expensive edge that would mean we have reduced the cost of an optimal solution.

Therefore, every edge bought due to the predictions costs at most OPT. □

Lemma 5.3.5. *The set A_2 can be partitioned into B_1 and B_2 so that $c(B_1) \leq OPT$ and $|B_2| \leq \eta$.*

Proof. Intuitively, B_1 is the set of *good* edges of $MST(\hat{R})$, while the rest are B_2 and can be thought of as *bad* edges, incurring a high total cost.

Suppose that the following procedure takes place. Starting with $S = MST(\hat{R})$ and $E' = MST(\hat{R}) \cap MST(\hat{R} \cap R)$, we iteratively pick an $e \in MST(\hat{R} \cap R) \setminus MST(\hat{R})$, add it to S and remove an edge $e' \in MST(\hat{R}) \setminus MST(\hat{R} \cap R)$. Then augment E' by the edge e'.

What this achieves is that $c(E') \leq c(MST(\hat{R} \cap R)) \leq OPT$, since E' started with the common edges between the two MSTs and only added the e' from $MST(\hat{R})$, each one of which is $c(e') \leq c(e)$ with $e \in MST(\hat{R} \cap R)$ using the same argument as in the previous lemma.

By choosing $B_1 = E' \cap A_2$ we immediately get $c(B_1) \leq OPT$ since $B_1 \subseteq E'$.

Then $B_2 = A_2 \setminus E'$. But $E' = k - \eta - 1$ since its edges are equal to those of $MST(\hat{R} \cap R)$. Also $|A_2| \leq k - 1$ since it only buys edges from $MST(\hat{R})$. Therefore $|B_2| \leq \eta$. □

As a whole, the competitive ratio is η, which means that the robustness of the algorithm is k, much worse than that of the online $O(\log k)$ competitive algorithm. Next, we will present a modification to the algorithm that fixes this problem.

$O(\log \eta)$-Competitive Algorithm. The previous algorithm did fairly well when it came to the mispredicted terminals. The problem was that when it bought a path from $MST(\hat{T})$ there was no guarrantee as to how much it might cost. Allowing for costs bounded by that of the least expensive single edge that connects the terminal with the other predictions is sufficient to solve this issue, since the cost will be once again comparable to that of the greedy algorithm.

Algorithm 8: Online Algorithm with Predicted Terminals

```
1  forall r_i in the input do
2      if r_i ∉ R̂ or r_i is the first predicted terminal to appear as input then
3          Buy the shortest path that connects r_i with R_{i-1}.
4      else if r_i ∈ R̂ then
5          P_i is the shortest path of MST(R̂) that connects r_i to R_{i-1} ∩ R̂.
6          e_i is the least expensive edge connecting r_i to R_{i-1} ∩ R̂
7          if ∃P'_i ⊆ P_i with cost c(P'_i) ∈ [c(e_i), 2c(e_i)] then
8              Buy P'_i.
9          if P'_i does not give a connected tree then
10             Buy e_i.
```

The lemmas from the previous algorithm apply here as well. What remains is bound A_2.

Lemma 5.3.6. *The cost of set A_2 is $c(A_2) \leq \log \eta \cdot OPT$.*

Proof. Set A_2 can be split into B_1 and B_2 as before.

Due to the range allowed as a cost for the path P'_i we obtain two bounds for the cost incurred in step i of the algorithm, denoted $\Delta_i c(A_2)$, as follows

$$\Delta_i c(A_2) \leq \min\{2c(P'_i), 3c(e_i)\}$$

Now we define *good* terminals, denoted $r_i \in T^G$, as those whose path bought by the algorithm is $P'_i \subseteq B_1$ and

$$\sum_{r_i \in T^G} \Delta_i c(A_2) \leq \sum_{r_i \in T^G} 2c(P'_i) \leq 2c(B_1) \leq O(1) \cdot OPT$$

The rest of them can be considered *bad* terminals, and we bound their cost by using the second bound

$$\sum_{r_i \in T^B} \Delta_i c(A_2) \leq \sum_{r_i \in T^G} 3c(e_i) \leq 3 \cdot \sum_{r_i \in T^G} c(e_i)$$

Notice that the sum is made up of the edges that would be bought by the greedy online algorithm if it was run on the $|B_2| = \eta$ terminals therefore

$$\sum_{r_i \in T^B} \Delta_i c(A_2) \leq O(\log \eta) \cdot OPT$$

□

Theorem 5.3.1. *The cost of the algorithm is $ALG \leq O(\log \eta) \cdot OPT$.*

This algorithm offers a significant improvement, as its robustness now matches the ratio of the greedy online algorithm, $O(\log k)$.

5.3.2 Using Metric Matching with Outliers as Error

In this section we discuss a more general approach to this problem. While outliers proved to be a useful error metric for Steiner Tree, using the framework of section 3.3, [Azar et al., 2021] showed that we can analyze the problem taking into account both the outliers and the specific distribution of the input and prediction in the induced metric space. Previously, the algorithms combined were a greedy online and an offline 2-approximation. As expected, the greedy online algorithm will still be used, but the offline algorithm will be swapped for one that solves the prize-collecting variant of the problem, still a 2-approximation.

Lemma 5.3.7. *The greedy online algorithm buying the shortest path that connects a new terminal $r_i \in R$ to the rest from [Imase and Waxman, 1991] that is $O(\log |R| + 2)$ competitive is also subset competitive.*

Proof. Suppose $R' \subseteq R$. We need to bound the cost of the online algorithm with input R that was incurred to connect the terminals of R'.

When request $r_i \in R'$ appears in the input, the closest terminal also in R' is r. If the algorithm decides to buy a different path P_i instead, that is because $c(P_i) \leq d(r_i, r)$, due to the terminals in $R \setminus R'$. But $d(r_i, r)$ is exactly the cost that would be paid if the algorithm was ran on R' only. Obviously, the total cost of running the online algorithm with input R' is $O(\log |R'| + 2) \cdot OPT'$, where obviously $OPT' \leq OPT$, therefore

$$ON(R') \leq O(\log |R'| + 2) \cdot OPT$$

□

Lemma 5.3.8. *There is a 2-competitive algorithm [Goemans and Williamson, 1995] for the prize collecting variation of Steiner Tree.*

We will assume that we are solving the rooted version of Steiner Tree, where one terminal, say ρ is the root and every other terminal needs to be connected to ρ. This is no different than the regular Steiner Tree, as any vertex can be considered a root, since all terminals will be connected in the tree.

Matching Error. The matching error will be calculated as a regular matching between some of the input R and the predicted $\hat{R}$, so as not to exceed D.

As for the analysis, we need to define the iteration i and bound $\hat{B}_{i-1}$ and ON_{m-1}, ON_m.

Iteration i. We would like this iteration to be the point at which the offline solutions bought, connect all the matched predictions in $\hat{T}$, where $|\hat{T}| = k$. This implies that $|\hat{R}| - k$ predicted terminals will not have been connected. Even though it is impossible to know whether the k connected predicted terminals will actually be the matched ones, it is sufficient to define i as the first major iteration when the lowest u selected by the algorithm is at most $|\hat{R}| - k$.

Theorem 5.3.2. *The cost incurred by the framework for Steiner Tree is*

$$ALG(R, |\hat{R}|) \leq O(\log(\min\{\Delta, |R|\} + 2)) \cdot OPT + O(1) \cdot OPT$$

Lemma 5.3.9. $\hat{B}_{i-1} \leq OPT + D$

Proof. From the definition of iteration i, we know that in the previous major iteration i' the predicted terminals not connected were $u_{i'} > |\hat{R}| - k$. Then

$$c(PARTIAL(\hat{R}, |\hat{R}| - k)) \geq c(PARTIAL(\hat{R}, u_{i'})) > 3\gamma \hat{B}_{i-1}.$$

otherwise iteration i would have been earlier.

From the analysis of the framework in Lemma 3.7, we know that $c(PARTIAL(\hat{R}, u_i)) \leq 3\gamma c(S^*)$. Then

$$c(PARTIAL(\hat{R}, |\hat{R}| - k)) \leq c(PARTIAL(\hat{R}, u_i)) \leq 3\gamma c(S^*) \leq 3\gamma OPT_{\hat{T}}$$

where the last inequality holds because S^* is the optimal solution on $\hat{R}$ that leaves $u_i \leq |\hat{R}| - k$ predicted terminals disconnected. $OPT_{\hat{T}}$ on the other hand, leaves disconnected exactly $\hat{R}| - |\hat{T}| = |\hat{R}| - k$ predicted terminals, therefore it a feasible solution and the bound follows.

Combining the two inequialities we have

$$\hat{B}_{i-1} \leq OPT_{\hat{T}}$$

At this point, we can use the matching cost cost to upper bound $OPT_{\hat{T}}$. If $r_i \in T$ is the matched terminal for every $\lambda(r_i) \in \hat{T}$ then we can buy an optimal solution on T denoted OPT_T, and connect all $\lambda(r_i)$ to their matched r_i and incur total cost $OPT_T + D$. Since $OPT_T \leq OPT$ we get the lemma.

□

Lemma 5.3.10. $ON_j \leq O(\log(\min\{|\Delta|, R\} + 2)) \cdot OPT + O(1) \cdot D$ *for* $j \in \{m-1, m\}$.

Proof. Let $j \in \{m-1, m\}$, and let $Q \subseteq R$ be the subsequence of requests considered in ON_j. Denote by $R' \subseteq R$ the subset of predicted requests that are satisfied by the $PARTIAL$ solution considered in iteration i (the solution whose facilities were opened at the end of iteration i).

The online algorithm ON_j operates on a modified input, in which the cost of a set of edges F_0 has been set to 0.

We partition Q into the following subsequences:

1. $Q_1 = Q \cap T$. We further partition Q_1 into the following sets:
   (a) $Q_{1,1} = \{r \in Q_1 | \lambda(r) \in R'\}$
   (b) $Q_{1,2} = \{r \in Q_1 | \lambda(r) \notin R \backslash R'\}$
2. $Q_2 = Q \backslash T$.

Intuitively, $Q_{1,1}$ is the part of T that appeared in iteration j and their matched predictions were covered by the offline solution on iteration i. Observe that $ON_j(Q) = ON_j(Q_{1,1}) + ON_j(Q_{1,2} \cup Q_2)$. We now bound each component separately.

Bound of $ON_j(Q_{1,1})$ For $r \in Q_{1,1}$, the matched prediction $\lambda(r)$ is covered by a path P with cost $c(P) \leq d(r, \rho)$ opened by PARTIAL in iteration i, since at most the algorithm will connect it all the way to the root. Therefore

$$\begin{aligned} ON_j(Q_{1,1}) &\leq \sum_{r \in Q_{1,1}} d(r, \rho) \\ &\leq \sum_{r \in Q_{1,1}} (d(r, \lambda(r)) + d(\lambda(r), \rho)) \end{aligned}$$

Observe that the path with distance $d(\lambda(r), \rho)$ has already been bought by the offline algorithm in order to connect $\lambda(r)$ and its cost for the online algorithm is 0. Then

$$ON_j(Q_{1,1}) \leq D$$

Bound of $ON_j(Q_{1,2} \cup Q_2)$ Using subset competitiveness, we have that

$$ON_j(Q_{1,2} \cup Q_2) \leq O(\log(|Q_{1,2} \cup Q_2| + 2)) \cdot OPT'$$

where OPT' is the optimal solution to Q with the cost of F_0 set to 0. Clearly, $OPT' \leq OPT$.

Now, observe that $|Q_{1,2}| \leq |\hat{R} \backslash \hat{R}'|$. From the definition of the major iteration i, and from Lemma 2.1 in the framework, it holds that $|\hat{R} \backslash \hat{R}'| \leq 2\gamma \cdot (|\hat{R}| - k) = 2\gamma |\hat{R} \backslash \hat{T}|$. As for Q_2, it holds that $Q_2 \leq |\hat{R} \backslash \hat{T}|$. These facts imply that

$$|Q_{1,2} \cup Q_2| \leq 2\gamma\Delta$$

In addition, it clearly holds that $|Q_{1,2} \cup Q_2| \leq |R|$. Therefore, it holds that

$$ON_j(Q_{1,2} \cup Q_2) \leq O(\log(min\{|R|, \Delta\} + 2)) \cdot OPT$$

Using all the previous bounds we get

$$ON_j(Q) \leq O(\log(min\{|R|, \Delta\} + 2)) \cdot OPT + O(1) \cdot D$$

□

The two Lemmas above suffice to prove the theorem using Lemmas 3.8, 3.9.

Notice that setting $D = 0$ we retrieve the same bound as in the previous section where we only considered outliers therefore the same robustness $O(\log |R|)$. As for consistency, using this algorithm we still obtain an $O(1)$ ratio. The difference is that the ratio degrades much more smoothly, as disturbing the positions of all terminals by only $\frac{\epsilon}{|R|}$ will give a bound of $O(1) \cdot OPT + \epsilon$ as opposed to the previous solution which would already have considered all predictions to be wrong, giving $O(\log |R|)$ ratio.

5.4 Online Facility Location

Facility Location is another well known NP-Hard network problem, where the objective is to cluster the input, not unlike k-means or k-median. On a high level, the input is made out of locations of clients on a map and we wish to buy facilities located as close to them as possible. In order to formalise the problem, assume a metric space $M(X, d)$, the set of possible demands (or clients) D and the set of possible locations for facilities F. Then, each $i \in F$ has a cost f_i and assigning a demand j to facility i incurs a cost equal to the distance $d(r_j, f_i)$. Notice how we abuse the notation so that f_i is both the cost and the location of the facility.

The integer program for this problem uses variables y_i to indicate whether a facility is open and x_{ij} to indicate whether demand j is assigned to facility i. The constraints are that each demand must be assigned to a facility and that this facility must be open in order to cover a demand.

$$\begin{aligned} \min & \sum_{i \in F} f_i \cdot y_i + \sum_{i,j \in F \times D} d(r_j, f_i) \cdot x_{ij} \\ s.t. & \sum_{i \in F} x_{ij} = 1 \quad \forall j \in D \\ & x_{ij} \leq y_i \quad \forall i \in F, j \in D \\ & x_{ij}, y_i \in \{0, 1\} \quad \forall i \in F, j \in D \end{aligned}$$

5.4.1 Prediction of the Input

Similarly with the case of Steiner Tree, we can use predictions of the locations of the demands, with the framework [Azar et al., 2021].

In this case, the online algorithm used will be the primal-dual from [Fotakis, 2011]. This algorithm is not subset competitive but the amortization that uses the variables $a(r_i) = 2 \cdot \min \{d(F_{i-1}, r_i), \min_{u \in V}(f_u - p_{i-1}(u) + d(u, r_i))\}$ of the solution is, where $p(u)$ is the potential calculated by the algorithm.

Lemma 5.4.1. *Online algorithm There is a $O(\log n)$-competitive online subset-competitive algorithm for Online Facility Location.*

The offline prize collecting algorithm used is [Xu and Xu, 2005].

Lemma 5.4.2. *There is a 1.8526-approximation algorithm for the prize-collecting version of Facility Location.*

The metric error used is simply a min cost matching between the input and the predictions. The analysis is identical to that of Steiner Tree, therefore we omit it and present the result.

Theorem 5.4.1. *The cost of the algorithm given by the framework is $ALG \leq O(\log(\min |R|, \Delta + 2)) \cdot OPT + O(1) \cdot D$.*

5.4.2 Multiple Predictions of the Solution

On the other hand, [Almanza et al., 2021] studies the setting where we have multiple predictions of the solution and we attempt to compare with the best combination of them. Formally, assume we are given sets $S_1, S_2, \ldots, S_k$ of facilities. Then, we wish to find an algorithm that incurs cost comparable to the optimal that uses only predicted facilities $S = \bigcup_i S_i$, $OPT(S)$.

Once we have that, we can combine it using the framework [Mahdian et al., 2012] with the optimal algorithm from [Fotakis, 2011] and achieve robustness.

Algorithm 9: Algorithm TakeHeed

Input: 2-HST family $(\mathcal{T}, D)$

```
Choose random HST T ~ D
forall p in the input do
    q_p ← arg min_{f∈S} d(f, p)
    Run PLUCK (T,root(T),q_p) and assign the facility returned to p
Function PLUCK(T,u,q)
    if u is a leaf then
        Open u if not already open
        return u
    if T(u) has no open leaf within distance f from q then
        w ← SELECTHEAVIESTCHILD (T,u,q)
        return PLUCK (T, w, q)
    else
        Let x be the child of u s.t. q is a leaf of T(x)
        if T(x) has an open leaf then
            // Find the open leaf
            return PLUCK (T, x, q)
        else
            Let l be the closest open leaf to q
            // Update potential
            p(u) ← p(u) + d_T(q, p)
            // Look for facility to open
            if p(u) ≥ f then PLUCK (T,x,q)
            else return l
Function SELECTHEAVIESTCHILD(T,u,q)
    return a child w of u s.t.
        1. T(w) has a leaf at distance no more than f/3 from q
        2. The number of leaves of T(w) is maximum (among those of the first condition)
```

Theorem 5.4.2. *There is an algorithm that given predictions $S_1, S_2, \ldots, S_k$ of the solution, incurs cost*

$$ALG \leq \min\{O(\log |S|) \cdot OPT(S), \frac{\log n}{\log \log n} \cdot OPT\}$$

The main technique used to obtain this result is the construction of Hierarchically Separated Trees for the metric space of predicted facilites S. This is possible due to the result of [Fakcharoenphol et al., 2004], which states that we can construct a 2-HST family $(\mathcal{T}D)$ for any metric space, where $\mathcal{T}$ is a set of HSTs and D is a probability distribution over them, since some of the properties are probabilistic.

The algorithm randomly chooses an HST $T \sim D$ and operates on that. It also represents each demand p of the input by its closest facility q_p in S. On each step, we need to select some open facility to which q_p will be assigned, or open a new one, which is done by calling PLUCK repeatedly. Function PLUCK is defined recursively and it traverses the tree in order to select the correct leaf (facility). If no leaf is open near q_p, then it has to select where to open a new one. It is in our best interest to open a new leaf both close to q_p and to as many

other demands as possible, which is achieved by function SelectHeaviestChild. On the other hand, if there are open leaves close to q_p we prefer to traverse in the direction of q if possible. If there is no such leaf in the subtree that contains q_p we will have to decide whether to open a new leaf or just assign the one we know is closest. But we should not allow too many leaves to open. Therefore, we delay the opening of a new leaf until enough potential has been accumulated in the internal nodes, at which point we allow the traversal towards that subtree.

The cost of this algorithm is $O(\log |S|) \cdot OPT(S)$. In the case of too many predictions, this cost can easily become too high, but it is important that the existence of even a few good facilities in each set can keep $OPT(S)$ close to OPT.

Chapter 6

Facility Leasing

In an effort to explore the strength of the framework [Azar et al., 2021] we apply it to a problem whose input evolves with time. The problem, Facility Leasing, first appeared in [Anthony and Gupta, 2007] and is essentially a variation of facility location in which demand locations change on each time step and need to be covered by facilities leased for limited amounts of time. As in Steiner Tree and Facility Location the algorithm will be buying solutions for part of the predicted demands in the form of facilities and their selected lease durations, and supplying them to the online algorithm to use if needed. The key element in the analysis is that of an error metric between instances, namely the predictions and the input, which separates the parts that need to be matched from those that can be considered outliers. While for time independent problems like Facility Location, bought facilities could potentially be used to cover any demand, in the leasing variation each leased facility can cover only demands that appear during its lease period and no others. This difficulty leads to the formulation of a nontrivial combinatorial optimization problem whose solution will directly express the error metric. We start by formally defining the problem. Then we present the two algorithms required by the framework and the remaining section proves the bounds achieved for the problem when predictions are given.

6.1 Online Facility Leasing

Overall the problem is split into T different time steps, not known prior to execution. We are given a set of potential locations for facilities F and another set of potential locations for demands D on a metric space (M, d). We are also given a list of K available lease durations, each denoted l_k, meaning that a lease l_k bought at time t for facility f_i will open a facility for the duration $[t, t + l_k)$, incurring a cost of f_i^k. The online input is split in sets $D_t \subseteq D$ each one indicating the positions of the demands for timestep t. In order to cover them, for all t, $r_{jt} \in D_t$ needs to be assigned to a facility f_i which is under lease for time t at a cost of $d(r_{jt}, f_i)$. Supposing that $\mathcal{F}$ is the set of all leased facilities in the form of (i, t, k), i.e. f_i^k starting at time t and $\mathcal{F}_t$ only those available during time t the total cost incurred is

$$\sum_{(i,t,k)\in\mathcal{F}} f_i^k + \sum_{(j,t):r_{jt}\in D_t} d(\mathcal{F}_t, r_{jt})$$

To formalise the study of the problem, we give the LP that corresponds to it as well as its dual.

Primal. The primal problem keeps track of all the costs incurred both for leasing and assigning demands to leased facilities, so that all demands are satisfied. The natural relaxation allows facilities to be fractionally leased and demands to be fractionally assigned to leased facilities.

$$
\begin{aligned}
\min \quad & \sum_{f=(i,t,k)\in\mathcal{F}} f_i^k \cdot y_f + \sum_{d\in\mathcal{D}} \sum_{f=(i,t',k)\in\mathcal{F}} d(f_i, r_d)\cdot x_{df} \\
s.t. \quad & \sum_{f=(i,t',k)\in\mathcal{F}: t\in[t',t'+l_k]} x_{df} \geq 1 \quad \forall d=(j,t)\in\mathcal{D} \\
& x_{df} \leq y_f \quad \forall d\in\mathcal{D}, f\in\mathcal{F} \\
& x_{df}, y_f \geq 0 \quad \forall d\in\mathcal{D}, f\in\mathcal{F}
\end{aligned}
$$

Dual. In the dual problem each demand $d=(j,t)$ declares how much it is willing to pay, a_d. The objective is to maximize the total payments. Each demand's payment should cover both its own assignment cost $d(f_i, r_d)$ and part of the lease cost f_i^k. In this sense we can consider $a_d - d(f_i, r_d)$ as the bid of demand r_d towards facility f_i at time t. The feasible solutions are those that include no overbidding by the demands for any given lease.

$$
\begin{aligned}
\max \quad & \sum_{d\in\mathcal{D}} a_d \\
s.t. \quad & \sum_{d=(j,t)\in\mathcal{D}, t\in I_{t'}^k} [a_d - d(f_i, r_d)]_+ \leq f_i^k \quad \forall (i,t',k)\in\mathcal{F} \\
& a_d \geq 0 \quad \forall d\in\mathcal{D}
\end{aligned}
$$

6.2 Subset Competitive online algorithm

First of all we need an online algorithm for the problem, with the added property of being subset competitive. This algorithm is given by Nagarajan, Williamson in [Nagarajan and Williamson, 2013] and will be denoted ON.

As the demand sets D_t arrive in an online fashion, the crucial decision to be made is whether we should assign a demand to an already leased facility, or lease a new one for it. In order to resolve this dilemma we use the dual LP and increase the dual variable a_{jt} corresponding to each demand (j,t), until it can pay for either the assignment cost to a previously leased open facility or contribute to the total bid towards a facility lease so that it reaches the lease cost.

Algorithm 10: Online Facility Leasing

Initialization
  Initialize $X \leftarrow \emptyset, X^k \leftarrow \emptyset, D \leftarrow \emptyset$
foreach $t \leq T$ **do**
  foreach $j \in D_t$ **do**
    Increase a_{jt} until one of the following occurs
      1. $a_{jt} = d(f_i, r_{jt})$ for some $(i,t^*,k)\in X$ and $t\in I_{t^*}^k$
      2. $[a_{jt} - d(f_i, r_{jt})]_+ + \sum_{(j',t')\in D, t'\in I_{t^*}^k} [\min\{a_{j't'}, d(X^k, r_{j't'})\} - d(f_{i'}, r_{j't'})]_+ = f_i^k$ for some $(i,t^*,k)\notin X$ and $t\in I_{t^*}^k$
    Assign (j,t) to the selected (i,t^*,k) and add it to D
    If it wasn't already open, add it to X and X^k

As in the case of Facility Location, in order to obtain subset competitiveness the cost of the algorithm is amortized using the variables of the LP. In the following suppose that R is made up of all the demands of the input.

Lemma 6.2.1. *The cost of the solution obtained by the algorithm is upper bounded by* $(K+1)\sum_{(j,t)\in R} a_{jt}$.

Therefore, the variables a_{jt} which correspond to the demands, multiplied by $K+1$ are a proper amortization of the algorithm's cost. The following lemma, whose result follows from Lemma 5.2 in their paper, shows that this amortization is in fact subset competitive, since their analysis is valid for any subset of the input that might be selected.

Lemma 6.2.2. *For any subset $R' \subseteq R$, the sum of dual variables corresponding to demands in R' is*

$$\sum_{(j,t)\in R'} a_{jt} \leq 2(H_{|R'|}+1)\cdot OPT$$

As a result, the amortized cost incurred due to any subset $R' \subseteq R$ is

$$(K+1)\sum_{(j,t)\in R'} a_{jt} \leq 2(K+1)(\log(|R'|+1)+1)\cdot OPT$$

6.3 Prize-collecting offline algorithm

In this case we pay a penalty for each demand that is not satisfied. The LP changes slightly to account for the penalties as follows.

Primal.

$$\begin{aligned}
\min \quad & \sum_{f=(i,t,k)\in\mathcal{F}} f_i^k \cdot y_f + \sum_{d\in\mathcal{D}} \sum_{f=(i,t',k)\in\mathcal{F}} d(f_i, r_d)\cdot x_{df} + \sum_{d\in\mathcal{D}} \pi_d \cdot z_d \\
s.t. \quad & z_d + \sum_{f=(i,t',k)\in\mathcal{F}: t\in[t',t'+l_k]} x_{df} \geq 1 \quad \forall d=(j,t)\in\mathcal{D} \\
& x_{df} \leq y_f \quad \forall d\in\mathcal{D}, f\in\mathcal{F} \\
& x_{df}, y_f, z_d \geq 0 \quad \forall d\in\mathcal{D}, f\in\mathcal{F}
\end{aligned}$$

Dual.

$$\begin{aligned}
\max \quad & \sum_{d\in\mathcal{D}} a_d \\
s.t. \quad & \sum_{d\in\mathcal{D}} [a_d - d(f_i, r_d)]_+ \leq f_i^k \quad \forall (i,t',k)\in\mathcal{F} \\
& a_d \leq \pi_d \quad \forall d\in\mathcal{D} \\
& a_d \geq 0 \quad \forall d\in\mathcal{D}
\end{aligned}$$

Now we can present a prize collecting offline algorithm with constant competitive ratio for facility leasing, denoted PC. This algorithm will solve the offline version of the problem where the whole input is known in advance, but any one of the demands r_{jt} can be left uncovered for a cost $\pi_{r_{jt}}$. This algorithm is given by Lima, San Felice, Lee in [de Lima et al., 2016]. We will say that demand r_{jt} *reaches* facility f_i when $a_{jt} \geq d(f_i, r_{jt})$.

The algorithm combines ideas from the online algorithm in [Nagarajan and Williamson, 2013] and the prize-collecting offline algorithm for Facility Location in [Charikar et al., 2001].

Algorithm 11: Prize-collecting algorithm for Facility Leasing

Initialization
  // X is the set of selected leased facilities and S is the set of remaining demands
  Initialize $X \leftarrow \emptyset, S \leftarrow \mathcal{D}$

while $S \neq \emptyset$ **do**
  Increase all dual variables a_d uniformly until one of the following happens
  if $\sum_{(j,t)\in\mathcal{D}} [a_{jt} - d(f_i, r_{jt})]_+ = f_i^k$ *for some lease* $(i,k,t) \in \mathcal{F}$ **then**
    Add $f = (i,k,t)$ to X
  else if $a_{jt} = d(f_i, r_{jt})$ *or* $a_{jt} = \pi_{jt}$ *for some* $(j,t) \in S$ *and some lease* $(i,k,t) \in X$ **then**
    Remove (j,t) from S

Create a graph G with
  $V[G] \leftarrow X$
  $E[G] \leftarrow \{(f_1, f_2) \in X^2 : \exists (j,t) \in \mathcal{D}$ that reaches both f_1 and f_2 $\}$
Find a maximal independent set $X' \subseteq X$ on G in decreasing order of lease duration
$\hat{X} \leftarrow \{(i,k,t-l_k), (i,k,t), (i,k,t+l_k) : (i,k,t) \in X'\}$
// Assign demands to leased facilities
foreach $(j,t) \in \mathcal{D}$ **do**
  if (j,t) *reaches some* $f \in X$ **then**
    Assign it to the closest leased facility in $\hat{X}$
  else
    Assign it to no facility

Specifically, it starts by uniformly increasing the dual variables until either some lease is paid for and consequently becomes available for demands to be assigned to it, or a demand pays for its assignment to an already leased facility, or a demand has paid enough for its own penalty. In the second and third cases, we can assume that the demand has paid as much as needed to be satisfied and its dual variable stops increasing.

At this point, it remains to be determined which facilities will actually be leased. By finding a maximal independent set we ensure that each demand *reaches* at most one facility in it. The interesting question is what happens to a demand r_d that reaches no facility in the independent set found. Due to the construction it must have reached some other facility f, which was excluded from the set as it had an edge with a facility f' in it, which means that these two facilities were both *reached* by another demand $r_{d'}$. Remember that in the construction of the set, the facilities were considered in decreasing order of their lease durations, therefore the facility f' will be open for a longer time than f would be and has some overlap with f in the durations of their leases. In order to ensure that the resulting solution is feasible it suffices to lease facility f' two more times with the same lease, before and after the lease selected by the independent set, which is exactly the reason the algorithm is 3-competitive.

Beyond this point, no more facilities are leased, it only remains to be decided for each demand whether it will be assigned to some open facility or pay the penalty.

Lemma 6.3.1. *PC is a prize-collecting algorithm for Facility Leasing achieving a constant ratio* $\gamma_{FL} = 3$.

6.4 Facility Leasing with Predicitons

Since the framework works in a black-box fashion, two algorithms are needed to be plugged into it. The algorithms PC and ON will be combined by the framework in order to utilize appropriately the predictions. As in the rest of the problems studied for the framework, we assume that an oracle provides a prediction $\hat{R}$ of the whole input in an offline fashion. Any solution on $\hat{R}$ bought with the purpose of being used to cover the actual input, has one caveat compared to Facility Location. It might be completely unusable as the leased facilities could correspond to lease periods during which there are no actual demands to be covered. As an example, a facility might be leased for a period that is too far in the future (or the past) from any demands of the actual input. This discrepancy will need to be treated appropriately by the error metric. Another issue is that of counting the cost of the online algorithm. From this point onwards it will be assumed that this cost is counted with respect to the amortization based on variables a_{jt} as described in Lemma 1.3. In fact, these are calculated in an online fashion for every set of demands D_t, which is needed since the offline solution is only bought when this online cost has doubled.

It still remains to be discussed what the error tuple (Δ, D) between two instances of the input, of outliers Δ and matching cost D will represent. Intuitively, it should capture the distances both in the spacial distribution of the demands and their temporal distribution, while not being too restrictive for either one of them. A naive approach would be to adapt the error as presented in Facility Location, by simply matching the two instances in every time step independently. It is easy to see why this approach fails catastrophically, for example in the case when the two instances are exactly the same but one is shifted by one time step relative to the other. In essence, any matching that doesn't take into account the specific clustering of demands in time, is bound to fail, in this exact sense. In order to overcome this obstacle we design a combinatorial optimization problem, which includes the optimal solution of one of the instances in its definition so that it can match only those demands that could possibly be matched. Of course, this is not enough yet, as the distributions of the one instance could be too far from the other to be able to be connected to any of its demands. But building on this idea, the next step is to also account for all the possible combinations of the extensions of the leases in this solution, each one of which helps measure the distance of the instances in terms of time.

Note that from now on we will drop the t from the demands r_{jt} for ease, assuming that all demands are ordered in some way. Any symbols denoted with a hat, like $\hat{r}_j$ correspond to the predicted instance, while the rest correspond to the actual input.

Error Metric. In order to match the input R to the predictions $\hat{R}$ we need to find subsets T and $\hat{T}$ respectively so that the optimal solution on T can be used to cover the predictions in $\hat{T}$, with extra cost D. This will be used later to upper bound the cost of $OPT_{\hat{T}}$.

Therefore we need to find a $T \subseteq R$ whose optimal solution is denoted OPT_T. The leases f_i^k bought by this solution can be extended in finitely (albeit exponentially) many combinations, without changing the assignments, each of them capturing the time distance between the instances and incurring an extra cost due to the extensions. The j-th of these solutions will be denoted $ExtOPT_T^{(j)}$.

At this point we need to constrain which demands can be matched with each other in any feasible solution. In order to achieve that, we construct a bipartite graph between T and $\hat{R}$ for each of the extension combinations in the following way:

- If a prediction $\hat{r}_j \in \hat{R}$ can be covered by some facilities in $ExtOPT_T^{(j)}$ (i.e. they are leased for a time period that includes t) then add all the edges between $\hat{r}_j$ and the demands that are assigned to these facilities only. The weight of these edges is the distance between the respective demands.

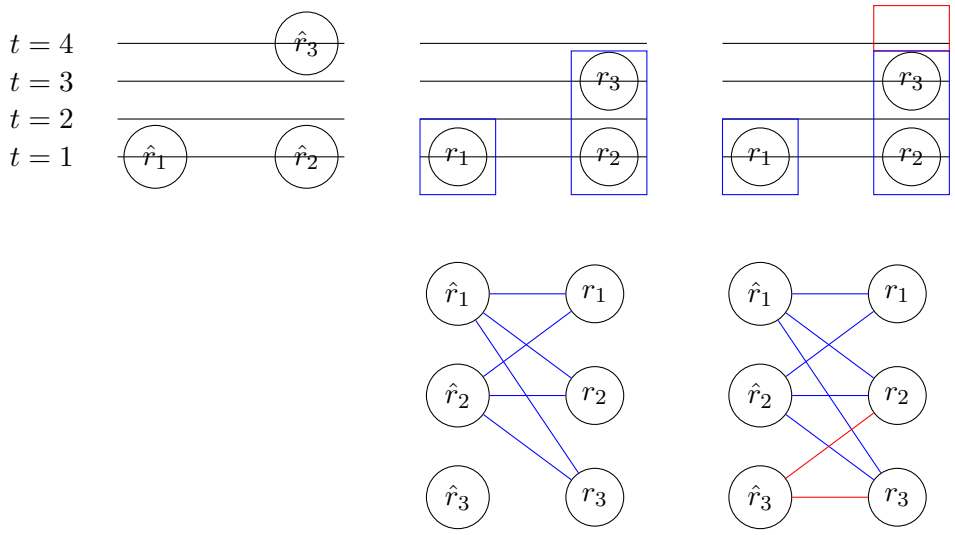

Figure 6.1: Example of matching where blue rectangles correspond to leased facilities of the optimal solution while red is a possible extension

On that graph we find a Bipartite Matching $M(T, \hat{R})$ so that

1. Its cost $M = c(M(T, \hat{R}))$ plus the cost of the extensions $E = c(ExtOPT_T^{(j)}) - c(OPT_T)$ is less than D.
2. Every $r_j \in T$ has an edge in the matching.

The second condition is necessary because T needs to be the actual matched subset of R. If it changes, the optimal solution also changes and the bipartite graph is no longer valid. In essence we pick one edge from each demand in T so that we have a low cost valid matching. The rest of the input and predictions, $R \setminus T \bigcup \hat{R} \setminus \hat{T}$ are the outliers Δ. As for the matching cost, intuitively E represents the distance of the two instances in terms of time, while M is their distance in terms of space. Both are necessary to define a reasonable matching. Of course, it is important to keep in mind that this problem is more of a theoretical formulation, not supposed to be solved efficiently. Nevertheless it is useful in the analysis due to the generality of the scenarios it covers.

As expected, for the different values of Δ and D, different solutions emerge. For instance, by choosing $\Delta = 0$, all of the input will have to be matched and there will be a minimum value of D for which this can be achieved. This value will correspond to the best combination of extensions which allow all demands to be covered. On the other hand, by choosing $D = 0$, all demands that are not situated in identical positions will be outliers. Note that a solution is always possible by choosing $T = \emptyset$. If it comes to that, the predictions are way too far from the demands in every way and we have prefered to discard them completely, making them all outliers. These simple cases show exactly how this error metric is capable of capturing every possible discrepancy between the instances.

At this point, we have defined every element necessary to state the main theorem of this section.

Theorem 6.4.1. *There is a learning augmented algorithm for Facility Leasing achieving the following bound, for every valid error tuple* (Δ, D)

$$Alg(R, \hat{R}) \leq O(K) \cdot D + O(K(\log(min\{|R|, \Delta\} + 1) + 1)) \cdot OPT$$

In order to prove the theorem, three ingredients are needed. The first is to define the iteration i which will split the execution of the framework in two parts. For the first part, it

suffices to bound the cost incurred by the algorithm until the previous major iteration, while for the second, it suffices to bound the cost of the online algorithm during the last iteration. The proofs following are similar to those given for Facility Location.

Iteration i. The framework [Azar et al., 2021] requires that an iteration i is selected, which splits the analysis in two parts. As in the rest of the problems, i is the first iteration in which u (approximately the predictions not covered by the offline solution) was chosen to be at most $|\hat{R}| - k$, where $k = |\hat{T}| = |T|$.

Lemma 6.4.1. *$\hat{B}_{i-1} \leq OPT + D$ where D is the error metric discussed above.*

Proof. If i' is the iteration when $B_{i'} = \hat{B}_{i-1}$, then $u_{i'} > |\hat{R}| - k$ and

$$PARTIAL(\hat{R}, |\hat{R}| - k) \geq 3\gamma \hat{B}_{i-1}$$

otherwise $u_{i'}$ would be smaller.

But $PARTIAL(\hat{R}, |\hat{R}| - k)$ also covers at least $k = |\hat{T}|$ predictions and by Lemma 2.1 it is upper bounded by the optimal solution on some k predictions, which is upper bounded by $OPT_{\hat{T}}$ since it fixes the k predictions. Therefore

$$PARTIAL(\hat{R}, |\hat{R}| - k) \leq 3\gamma c(OPT_{\hat{T}})$$

As a result

$$\hat{B}_{i-1} \leq c(OPT_{\hat{T}})$$

It is sufficient to bound $c(OPT_{\hat{T}})$ by showing how $\hat{T}$ can be covered using the facilities of OPT_T whose cost is bounded by that of OPT. By the definition of the error metric, the lease-extended facilities of $ExtOPT_T$ (where the index j was dropped to indicate the combination that was chosen) can cover $\hat{T}$ as follows

$$\begin{aligned}
\hat{B}_{i-1} \leq c(OPT_{\hat{T}}) &\leq \sum_{r_j \in T} d(F, \lambda(r_j)) + Facility(OPT_T) + E \\
&\leq \sum_{r_j \in T} d(\lambda(r_j), r_j) + \sum_{r_j \in T} d(F, r_j) + Facility(OPT_T) + E \\
&\leq M + OPT_T + E \\
&\leq OPT + D
\end{aligned}$$

by triangle inequality, after properly extending the leases. □

Lemma 6.4.2. *It holds that*

$$\max\{ON_{m-1}, ON_m\} \leq O(K)D + O(K(log(min\{|R|, \Delta\} + 1) + 1)) \cdot OPT$$

Proof. This proof follows from that of Lemma 3.5 in [Azar et al., 2021] with changes only in the bound of $ON_j(Q_{1,1})$ and is given for ease of the reader.

Let $j \in \{m-1, m\}$, and let $Q \subseteq R$ be the subsequence of requests considered in ON_j. Denote by $R' \subseteq R$ the subset of predicted requests that are satisfied by the $PARTIAL$ solution considered in iteration i (the solution whose facilities were opened at the end of iteration i).

The online algorithm ON_j operates on a modified input, in which the cost of a set of facilities F_0 is set to 0.

We partition Q into the following subsequences:

1. $Q_1 = Q \cap T$. We further partition Q_1 into the following sets:

(a) $Q_{1,1} = \{r \in Q_1 | \lambda(r) \in R'\}$

(b) $Q_{1,2} = \{r \in Q_1 | \lambda(r) \notin R \backslash R'\}$

2. $Q_2 = Q \backslash T$.

Intuitively, $Q_{1,1}$ is the part of T that appeared in iteration j and their matched predictions were covered by the offline solution on iteration i. Observe that $ON_j(Q) = ON_j(Q_{1,1}) + ON_j(Q_{1,2} \cup Q_2)$. We now bound each component separately.

Bound of $ON_j(Q_{1,1})$ For $r = (j,t) \in Q_{1,1}$, the matched prediction $\lambda(r)$ is covered by a facility $c_{\lambda(r)}$ opened by PARTIAL in iteration i. The online algorithm will stop increasing $a_r = a_{jt}$ at some point before it reaches $d(c_{\lambda(r)}, r)$ since it belongs to the set of opened facilities X. This implies $a_r \leq d(c_{\lambda(r)}, r)$. Using the bound for the cost

$$\begin{aligned} ON_j(Q_{1,1}) &\leq (K+1) \sum_r d(c_{\lambda(r)}, r) \\ &\leq (K+1) \sum_r \left(d(c_{\lambda(r)}, \lambda(r)) + d(\lambda(r), r) \right) \\ &\leq (K+1) \sum_r d(c_{\hat{r}}, \hat{r}) + (K+1)D \\ &\leq (K+1) \sum_r d(F_0, \hat{r}) + (K+1)D \end{aligned}$$

where $d(F_0, \hat{r}) \leq 3\gamma(2\hat{B}_{i-1} + O(1))D$ as in the proof of the Lemma and $ON_j(Q_{1,1}) \leq O(K)D + O(K)OPT$

Bound of $ON_j(Q_{1,2} \cup Q_2)$ Using subset competitiveness, we have that

$$ON_j(Q_{1,2} \cup Q_2) \leq O(\log(|Q_{1,2} \cup Q_2| + 1) + 1)) \cdot OPT'$$

where OPT' is the optimal solution to Q with the cost of F_0 set to 0. Clearly, $OPT' \leq OPT$.

Now, observe that $|Q_{1,2}| \leq |\hat{R} \backslash \hat{R}'|$. From the definition of the major iteration i, and from Lemma 2.1 in the framework, it holds that $|\hat{R} \backslash \hat{R}'| \leq 2\gamma \cdot (|\hat{R}| - k) = 2\gamma |\hat{R} \backslash \hat{T}|$. As for Q_2, it holds that $Q_2 \leq |\hat{R} \backslash \hat{T}|$. These facts imply that

$$|Q_{1,2} \cup Q_2| \leq 2\gamma\Delta$$

In addition, it clearly holds that $|Q_{1,2} \cup Q_2| \leq |R|$. Therefore, it holds that

$$ON_j(Q_{1,2} \cup Q_2) \leq O(K(\log(min\{|R|, \Delta\} + 1) + 1)) \cdot OPT$$

Using all the previous bounds we get

$$ON_j \leq O(K) \cdot D + O(K(\log(min\{|R|, \Delta\} + 1) + 1)) \cdot OPT$$

□

The two lemmas prove the theorem.

Chapter 7

Conclusions

In this thesis we explored the field of Learning-Augmented algorithms for the category of Online problems. In essence, the problem lies in our inability to evaluate the quality of predictions during runtime. Therefore, we need to design algorithms that both explore the additional information and discard it when it proves untrustworthy. When the predictions appear in an online fashion, it is natural to attempt to combine different online algorithms in order to remain competitive with the best among them. In these cases we typically attempt to design an algorithm that trusts the predictions in some way and then combine it with the best known online algorithm without predictions. When we are given predictions in an offline fashion, the natural approach is to combine an online algorithm with an offline algorithm, each handling the input and the predictions respectively. It is also common to modify standard methods, like Primal-Dual, in order to take advantage of the predictions.

Focusing on Network Problems, we studied a framework that elegantly achieves to separate good from bad predictions in an online fashion and applied it to Online Facility Leasing, by providing a way to measure the extra cost these predictions will force this algorithm to incur.

The importance of this result is two-fold.

- On one hand, the problem itself had not been studied in the setting of predictions, therefore the bound we obtained is the first for it. It is expected that it can be improved, either by obtaining better competitive ratios for the online version, or by designing a problem specific algorithm that takes even better advantage of predictions.
- On the other hand, we can deduce that time dependency is a feature that in some cases can be handled by known methods. This motivates the study of further problems whose input evolves with time in the setting of predictions.

Such problems were initiated in [Anthony and Gupta, 2007] and are essentially Leasing versions of known NP-Hard problems. In the same way that Facility Leasing replaces the purchase of facilities with the ability to lease them, Leasing Steiner Tree replaces the purchase of edges with the ability to lease them, as the distribution of terminals changes. Other such variants of problems are Leasing Vertex Cover and Leasing Set Cover with similar formulations.

Another exciting direction is that of Multistage Matroid Maintenance (MMM). Intuitively it can be thought of as maintaining a Spanning Tree with edges that each can only be bought for a certain interval. Ultimately, a goal would be to improve the current results for Metrical Task Systems (MTS), as this problem genelarizes MMM.

Bibliography

[Agrawal et al., 2022] Agrawal, P., Balkanski, E., Gkatzelis, V., Ou, T., and Tan, X. (2022). Learning-augmented mechanism design: Leveraging predictions for facility location.

[Almanza et al., 2021] Almanza, M., Chierichetti, F., Lattanzi, S., Panconesi, A., and Re, G. (2021). Online facility location with multiple advice. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 4661–4673. Curran Associates, Inc.

[Alon et al., 2003] Alon, N., Awerbuch, B., and Azar, Y. (2003). The online set cover problem. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, pages 100–105, New York, NY, USA. Association for Computing Machinery.

[Anand et al., 2022a] Anand, K., Ge, R., Kumar, A., and Panigrahi, D. (2022a). Online algorithms with multiple predictions.

[Anand et al., 2022b] Anand, K., Ge, R., Kumar, A., and Panigrahi, D. (2022b). A regression approach to learning-augmented online algorithms.

[Anand et al., 2022c] Anand, K., Ge, R., and Panigrahi, D. (2022c). Customizing ml predictions for online algorithms.

[Anthony and Gupta, 2007] Anthony, B. M. and Gupta, A. (2007). Infrastructure leasing problems. In Fischetti, M. and Williamson, D. P., editors, *Integer Programming and Combinatorial Optimization*, pages 424–438, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Antoniadis et al., 2020] Antoniadis, A., Coester, C., Elias, M., Polak, A., and Simon, B. (2020). Online metric algorithms with untrusted predictions.

[Azar et al., 2021] Azar, Y., Panigrahi, D., and Touitou, N. (2021). Online graph algorithms with predictions. *CoRR*, abs/2112.11831.

[Balkanski et al., 2022] Balkanski, E., Gkatzelis, V., and Tan, X. (2022). Strategyproof scheduling with predictions.

[Bamas et al., 2020] Bamas, É., Maggiori, A., and Svensson, O. (2020). The primal-dual method for learning augmented algorithms. *CoRR*, abs/2010.11632.

[Bansal et al., 2020] Bansal, N., Coester, C., Kumar, R., Purohit, M., and Vee, E. (2020). Learning-augmented weighted paging.

[Belady, 1966] Belady, L. A. (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101.

[Bernardini et al., 2022] Bernardini, G., Lindermayr, A., Marchetti-Spaccamela, A., Megow, N., Stougie, L., and Sweering, M. (2022). A universal error measure for input predictions applied to online graph problems.

[Boyar et al., 2022] Boyar, J., Favrholdt, L. M., and Larsen, K. S. (2022). Online unit profit knapsack with untrusted predictions.

[Charikar et al., 2001] Charikar, M., Khuller, S., Mount, D. M., and Narasimhan, G. (2001). Algorithms for facility location problems with outliers. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 642–651, USA. Society for Industrial and Applied Mathematics.

[Cho et al., 2022] Cho, W.-H., Henderson, S., and Shmoys, D. (2022). Scheduling with predictions.

[de Lima et al., 2016] de Lima, M. S., Felice, M. C. S., and Lee, O. (2016). Facility leasing with penalties. *CoRR*, abs/1610.00575.

[Dinitz et al., 2021] Dinitz, M., Im, S., Lavastida, T., Moseley, B., and Vassilvitskii, S. (2021). Faster matchings via learned duals.

[Du et al., 2021] Du, E., Wang, F., and Mitzenmacher, M. (2021). Putting the "learning" into learning-augmented algorithms for frequency estimation. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2860–2869. PMLR.

[Fakcharoenphol et al., 2004] Fakcharoenphol, J., Rao, S., and Talwar, K. (2004). A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497. Special Issue on STOC 2003.

[Feijen and Schäfer, 2021] Feijen, W. and Schäfer, G. (2021). Using machine learning predictions to speed-up dijkstra's shortest path algorithm.

[Fiat et al., 1991] Fiat, A., Karp, R. M., Luby, M., McGeoch, L. A., Sleator, D. D., and Young, N. E. (1991). Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699.

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

[Fotakis et al., 2021] Fotakis, D., Gergatsouli, E., Gouleakis, T., and Patris, N. (2021). Learning augmented online facility location. *CoRR*, abs/2107.08277.

[Gkatzelis et al., 2022] Gkatzelis, V., Kollias, K., Sgouritsa, A., and Tan, X. (2022). Improved price of anarchy via predictions.

[Goemans and Williamson, 1995] Goemans, M. X. and Williamson, D. P. (1995). A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317.

[Gollapudi and Panigrahi, 2019] Gollapudi, S. and Panigrahi, D. (2019). Online algorithms for rent-or-buy with expert advice. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2319–2327. PMLR.

[Gupta and Nagarajan, 2012] Gupta, A. and Nagarajan, V. (2012). Approximating sparse covering integer programs online. *CoRR*, abs/1205.0175.

[Im et al., 2021a] Im, S., Kumar, R., Montazer Qaem, M., and Purohit, M. (2021a). Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 285–294, New York, NY, USA. Association for Computing Machinery.

[Im et al., 2021b] Im, S., Kumar, R., Montazer Qaem, M., and Purohit, M. (2021b). Online knapsack with frequency predictions. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 2733–2743. Curran Associates, Inc.

[Imase and Waxman, 1991] Imase, M. and Waxman, B. M. (1991). Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384.

[Istrate and Bonchis, 2022] Istrate, G. and Bonchis, C. (2022). Mechanism design with predictions for obnoxious facility location.

[Jiang et al., 2021] Jiang, S. H., Liu, E., Lyu, Y., Tang, Z. G., and Zhang, Y. (2021). Online facility location with predictions. *CoRR*, abs/2110.08840.

[Jiang et al., 2020] Jiang, Z., Panigrahi, D., and Sun, K. (2020). Online algorithms for weighted paging with predictions.

[Karlin et al., 1990] Karlin, A. R., Manasse, M. S., McGeoch, L. A., and Owicki, S. (1990). Competitive randomized algorithms for non-uniform problems. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 301–309, USA. Society for Industrial and Applied Mathematics.

[Karp, 1972] Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

[Khodak et al., 2022] Khodak, M., Balcan, M.-F., Talwalkar, A., and Vassilvitskii, S. (2022). Learning predictions for algorithms with predictions.

[Kou et al., 1981] Kou, L., Markowsky, G., and Berman, L. (1981). A fast algorithm for steiner trees. *Acta Informatica*, 15:141–145.

[Lindermayr and Megow, 2022] Lindermayr, A. and Megow, N. (2022). Permutation predictions for non-clairvoyant scheduling.

[Lu et al., 2020] Lu, P., Ren, X., Sun, E., and Zhang, Y. (2020). Generalized sorting with predictions.

[Lykouris and Vassilvitskii, 2018] Lykouris, T. and Vassilvitskii, S. (2018). Competitive caching with machine learned advice. *CoRR*, abs/1802.05399.

[Mahdian et al., 2012] Mahdian, M., Nazerzadeh, H., and Saberi, A. (2012). Online optimization with uncertain information. *ACM Trans. Algorithms*, 8(1).

[Meyerson, 2001] Meyerson, A. (2001). Online facility location. In *Foundations of Computer Science*, pages 426– 431.

[Meyerson, 2005] Meyerson, A. (2005). The parking permit problem. In *Foundations of Computer Science*, pages 274 – 282.

[Mitzenmacher, 2019] Mitzenmacher, M. (2019). Scheduling with predictions and the price of misprediction.

[Motwani et al., 1994] Motwani, R., Phillips, S., and Torng, E. (1994). Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47.

[Nagarajan and Williamson, 2013] Nagarajan, C. and Williamson, D. P. (2013). Offline and online facility leasing. *Discrete Optimization*, 10(4):361–370.

[Polak and Zub, 2022] Polak, A. and Zub, M. (2022). Learning-augmented maximum flow.

[Purohit et al., 2018] Purohit, M., Svitkina, Z., and Kumar, R. (2018). Improving online algorithms via ml predictions. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Raghavan, 1988] Raghavan, P. (1988). Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143.

[Rohatgi, 2019] Rohatgi, D. (2019). Near-optimal bounds for online caching with machine learned advice.

[Thang and Durr, 2021] Thang, N. K. and Durr, C. (2021). Online primal-dual algorithms with predictions for packing problems.

[Wei, 2020] Wei, A. (2020). Better and simpler learning-augmented online caching.

[Wei and Zhang, 2020] Wei, A. and Zhang, F. (2020). Optimal robustness-consistency tradeoffs for learning-augmented online algorithms.

[Xu and Lu, 2022] Xu, C. and Lu, P. (2022). Mechanism design with predictions.

[Xu and Moseley, 2021] Xu, C. and Moseley, B. (2021). Learning-augmented algorithms for online steiner tree. *CoRR*, abs/2112.05353.

[Xu and Zhang, 2022] Xu, C. and Zhang, G. (2022). Learning-augmented algorithms for online subset sum.

[Xu and Xu, 2005] Xu, G. and Xu, J. (2005). An improved approximation algorithm for uncapacitated facility location problem with penalties. In Wang, L., editor, *Computing and Combinatorics*, pages 644–653, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Zhao et al., 2022] Zhao, T., Li, W., and Zomaya, A. Y. (2022). Uniform machine scheduling with predictions. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32(1):413–422.