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

Online αλγόριθμοι με προβλέψεις

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΟΝΑΣ ΝΙΚΟΛΑΟΥ

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

Αθήνα, Νοέμβριος 2021

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

Online αλγόριθμοι με προβλέψεις

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΑΣΟΝΑΣ ΝΙΚΟΛΑΟΥ

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15η Νοεμβρίου 2021.

Δημήτριος Φωτάκης Αν. Καθηγητής Ε.Μ.Π.	Αριστείδης Παγουρτζής Καθηγητής Ε.Μ.Π.	Χρήστος Τζάμος Αν. Καθηγητής Wisconsin-Madison

Αθήνα, Νοέμβριος 2021

Ιάσονας Νικολάου

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

Περίληψη

Στην παρούσα διπλωματική εργασία μελετάμε online αλγορίθμους με προβλέψεις (learning-augmented online algorithms), δηλαδή, online αλγορίθμους οι οποίοι έχουν πρόσβαση σε ένα μαντείο που κάνει προβλέψεις για το μέλλον. Παρουσιάζουμε τα πιο θεμελιώδη αποτελέσματα τού τομέα καθώς και τις βασικές αρχές που διέπουν την ανάλυση online αλγορίθμων με προβλέψεις. Επιπλέον, εξετάζουμε το πρόβλημα Multistage Matroid Maintenance (MMM) και σχεδιάζουμε online αλγορίθμους με προβλέψεις οι οποίοι έχουν καλύτερη επίδοση από αυτή των κλασικών online αλγορίθμων.

Στο πρόβλημα MMM καλούμαστε να διατηρήσουμε ένα spanning tree σε ένα δυναμικό γράφημα στο οποίο οι ακμές είναι διαθέσιμες μόνο για συγκεκριμένες χρονικές στιγμές. Στην learning-augmented εκδοχή είμαστε εξοπλισμένοι με ένα μαντείο το οποίο προβλέπει τον χρόνο ζωής κάθε ακμής. Στόχος είναι να αγοράσουμε συνολικά όσο το δυνατόν λιγότερες ακμές.

Αναλύουμε τον αλγόριθμο που ακολουθεί τυφλά τις προβλέψεις και δείχνουμε ότι αν οι προβλέψεις είναι ακριβείς ο αλγόριθμος ξεπερνά την απόδοση των κλασικών online αλγορίθμων. Αντίθετα, αν οι προβλέψεις είναι ανακριβείς η απόδοση τού αλγορίθμου είναι εφάμιλλη με αυτή τού βέλτιστου online αλγορίθμου.

Λέξεις κλειδιά

Online αλγόριθμοι, Online αλγόριθμοι με προβλέψεις

Abstract

In this thesis we study learning-augmented online algorithms. That is, online algorithms that have access to an oracle making predictions about the future. We present the most fundamental results of the field and explain the basic principles of learning-augmented analysis. Furthermore, we examine the Multistage Matroid Maintenance (MMM) problem and design learning-augmented algorithms that improve the bounds achieved via classical online algorithms. The Multistage Matroid Maintenance problem describes settings where we have to maintain a base of a matroid while the underlying costs are changing. We define a framework to incorporate predictions. Then, we analyze the algorithm blindly following the predictions and show that if the predictions are accurate our algorithm outperforms classical online algorithms. If the predictions are inaccurate the performance of our algorithm falls back to that of the optimal online algorithm.

Key words

Online algorithms, Learning-augmented Online algorithms, Algorithms with predictions

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή αυτής της διατριβής, κ. Δημήτρη Φωτάκη, για τη συνεχή καθοδήγηση και εμπιστοσύνη του. Θέλω να ευχαριστήσω ακόμα τον συμφοιτητή Δημήτρη Χρήστου, ο οποίος με βοήθησε στα αρχικά στάδια αυτής της εργασίας. Θα ήθελα τέλος να ευχαριστήσω την οικογένειά μου, η οποία με υποστήριξε και έκανε δυνατή την απερίσπαστη ενασχόλησή μου τόσο με την εκπόνηση της διπλωματικής μου, όσο και συνολικά με τις σπουδές μου.

Ιάσονας Νικολάου,

Αθήνα, 15η Νοεμβρίου 2021

Περιεχόμενα

Εκτεταμένη Ελληνική Περίληψη

Το βασικό σκέλος της παρούσας διπλωματικής εργασίας έχει αποδωθεί στην αγγλική γλώσσα, κυρίως για λόγους προσβασιμότητας. Σε αυτό το κομμάτι της, συνοψίζουμε το περιεχόμενό της, δίνοντας έμφαση στους βασικούς ορισμούς, τις μεθοδολογίες και τα θεωρήματα, αλλά χωρίς τις μαθηματικές αποδείξεις. Η δομή της ενότητας αυτής είναι σε ένα προς ένα αντιστοίχηση με το (αγγλικό) κείμενο της διπλωματικής εργασίας.

Εισαγωγή

Σε ένα κλασικό πρόβλημα βελτιστοποίησης μας δίνεται ως είσοδος ένα συγκεκριμένο στιγμιότυπο ενός προβλήματος και ζητείται να υπολογιστεί μία βέλτιστη λύση. Ωστόσο, σε πολλά πρακτικά προβλήματα δεν είναι γνωστή ολόκληρη η είσοδος εξ αρχής αλλά αποκαλύπτεται σταδιακά. Ένα τέτοιο πρόβλημα είναι ο υπολογισμός της ελάχιστης διαδρομής μεταξύ δύο πόλεων ενόσω η κίνηση στους δρόμους αλλάζει, ατυχήματα που συμβαίνουν καθυστερούν την κίνηση, δρόμοι κλείνουν για διάφορους λόγους, κ.λπ. Σε ένα τέτοιο πρόβλημα, πρέπει συνεχώς να προσαρμόζομαστε στην νέα πληροφορία που είναι διαθέσιμη και να λαμβάνουμε αποφάσεις βασιζόμενοι σε ελλιπείς πληροφορίες. Τέτοια προβλήματα ονομάζονται online προβλήματα και οι αλγόριθμοι που τα λύνουν ονομάζονται online αλγόριθμοι. Γνωστά online προβλήματα είναι το paging problem [64], το job-scheduling problem [21], το portfolio selection problem [29], και πολλά ακόμη [19]. Τα online προβλήματα έχουν ποικίλες εφαρμογές σε διάφορους τομείς όπως η πληροφορική, τα οικονομικά και η επιχειρησιακή έρευνα.

Το ερώτημα που ανακύπτει είναι το ακόλουθο. Πώς σχεδιάζουμε, αναλύουμε και αξιολογούμε online αλγορίθμους. Η παραδοσιακή απάντηση στο ερώτημα είναι μέσω της *competitive analysis*. Συγκεκριμένα, συγκρίνουμε την επίδοση ενός online αλγορίθμου με εκείνη του βέλτιστου (offline) αλγορίθμου που έχει γνώση ολόκληρης της εισόδου.

As θεωρήσουμε ένα πρόβλημα ελαχιστοποίησης. Σχεδιάζουμε έναν online αλγόριθμο ALG για το πρόβλημα, με κόστος το πολύ C φορές αυτό του βέλτιστου αλγορίθμου OPT για κάθε στιγμιότυπο του προβλήματος. Δηλαδή, $ALG \leq c \cdot OPT$, για κάθε στιγμιότυπο του προβλήματος. Λέμε ότι ο αλγόριθμος ALG είναι c-competitive ή διαφορετικά ότι το *competitive ratio* του ALG είναι c.

Η competitive analysis έχει χρησιμοποιηθεί για την ανάλυση και την σχεδίαση πολλών online αλγορίθμων, ωστόσο έχει κάποιες αδυναμίες. Πιο συγκεκριμένα, είναι υπερβολικά περιοριστική εφόσον απαιτεί την σύγκριση ενός online αλγορίθμου με τον βέλτιστο offline αλγόριθμο. Για πολλά προβλήματα υπάρχουν Online αλγόριθμοι οι οποίοι έχουν καλή επίδοση στην πλειονότητα των στιγμιοτύπων, αλλά σε κάποια εξαιρετικά παθολογικά στιγμιότυπα δεν αποδίδουν καλά. Ωστόσο, στην competitive analysis μας ενδιαφέρει η χειρότερη περίπτωση και έτσι οι αλγόριθμοι αυτοί παρουσιάζονται ως μη αποδοτικοί. Για τον λόγο αυτό είναι δύσκολο να συγκρίνουμε online αλγοριθμούς. Συχνά, αλγόριθμοι που έχουν το ίδιο competitive ratio αποδίδουν πολύ διαφορετικά στην πράξη.

Για να ξεπεραστούν οι παραπάνω αδυναμίες έχουν προταθεί πολλές εναλλακτικές. Πριν τις αναφέρουμε ας δούμε την κύρια αιτία των αδυναμιών. Η βασική αιτία είναι ότι σε προβλήματα του πραγματικού κόσμου η είσοδος σπανίως είναι η χειρότερη δυνατή ή τελείως τυχαία. Στην πράξη η είσοδος ακολουθεί συγκεκριμένα μοτίβα που είναι δυνατόν να προβλεφθούν και τα οποία εκμεταλλεύονται οι εμπειρικά αποδοτικοί αλγόριθμοι. Για τον λόγο αυτό έχουν δημιουργηθεί

οι τομείς beyond-competitive analysis [46] και beyond-worst-case analysis [61]. Στους τομείς αυτούς ανήκουν και οι *learning-augmented online algorithms* που είναι το θέμα της παρούσας διπλωματικής εργασίας.

Συγκεκριμένα, μελετάμε τον σχεδιασμό online αλγορίθμων, οι οποίοι έχουν πρόσβαση σε ένα μαντείο που κάνει προβλέψεις για την μελοντική είσοδο. Βεβαίως, οι προβλέψεις μπορεί να μην είναι ακριβείς. Γι'αυτό επιθυμούμε τον σχεδιασμό αλγορίθμων, οι οποίοι θα εκμεταλλεύονται τις προβλέψεις, όταν αυτές είναι ακριβείς, που θα βελτιώνουν το competitive ratio των παραδοσιακών online αλγορίθμων. Όταν οι προβλέψεις είναι ανακριβείς θα πρέπει ο αλγόριθμός μας να έχει εφάμιλλη επίδοση με εκείνη των online αλγορίθμων χωρίς προβλέψεις.

Η ιδέα των learning-augmented online αλγορίθμων εισήχθη από τους Lykouris και Vassilvtiskii [51] όταν έλυσαν το paging problem χρησιμοποιώντας προβλέψεις. Από τότε έχει δημοσιευτεί σημαντικός όγκος εργασιών σχετικά με το θέμα. Ενδεικτικά έχουν εξεταστεί τα ακόλουθα προβλήματα: το ski rental problem [49], scheduling [37], metrical task systems [9], secretary problem [10] και πολλά άλλα. Τέλος, αλγόριθμοι με προβλέψεις έχουν χρησιμοποιηθεί επιτυχώς και στην περιοχή των δομών δεδομένων [47, 55].

Στην παρούσα εργασία εκτός από την παρουσίαση των πιο θεμελειωδών αποτελεσμάτων του τομέα, θα μελετήσουμε και το πρόβλημα *Multistage Matroid Maintenance (MMM) problem*. Το πρόβλημα αυτό μελετήθηκε (χωρίς τις προβλέψεις) από τους Gupta, Talwar, Wieder [35].

Στα περισσότερα προβλήματα συνδυαστικής βελτιστοποίησης ασχολούμαστε με την εύρεση της βέλτιστης λύσης για ένα στιγμιότυπο "παγωμένο" στον χρόνο. Ωστόσο, στον πραγματικό κόσμο πολλές φορές καλούμαστε να λύσουμε ένα πρόβλημα ξανά και ξανά, ενώ η συνάρτηση κόστους αλλάζει κάθε στιγμή. Μία προσέγγιση είναι να λύνουμε το πρόβλημα κάθε φορά από την αρχή, Συνήθως, όμως, η μετάβαση από μία λύση σε μία άλλη επιφέρει κάποιο κόστος. Μία άλλη προσέγγιση είναι να προσαρμόσουμε την παλιά λύση στο νέο στιγμιότυπο. Ένας συνδυασμός των παραπάνω προσεγγίσεων είναι συνήθως η βέλτιστη στρατηγική.

Στο Multistage Matroid Maintenance (MMM) problem στόχος είναι να διατηρούμε κάθε χρονική στιγμή μία βάση ενός matroid. Θα ορίσουμε το matroid στο κεφάλαιο 8, αλλά για τώρα αρκεί να σκεφτόμαστε μία βάση ενός matroid ως ένα συνδετικό δέντρο ενός γραφήματος. Οι ακμές του γραφήματος είναι διαθέσιμες για συγκεκριμένες χρονικές στιγμές και πρέπει να διατηρούμε ένα συνδετικό δέντρο σε κάθε χρονική στιγμή. Στόχος είναι η ελαχιστοποίηση του συνολικού κόστους.

Οργάνωση. Στο κεφάλαιο 2, δίνουμε τις βασικούς ορισμούς στο πεδίο των online αλγορίθμων και κάνουμε μια σύντομη επισκόπηση δύο κλασικών online προβλημάτων - paging problem και metrical task system problem. Τα κεφάλαια 3-7 αποτελούν μια εκτεταμένη παρουσίαση των πιο βασικών αποτελεσμάτων του τομέα των learning-augmented online αλγορίθμων. Στο κεφάλαιο 3, εισαγάγουμε τις βασικές έννοιες της learning-augmented ανάλυσης εξετάζοντας το Ski rental problem και το non-clairvoyant job-scheduling πρόβλημα. Στο κεφάλαιο 4, εξετάζουμε το paging problem με προβλέψεις, το οποίο είναι και το πρώτο που λύθηκε υπό το πρίσμα της learning-augmented ανάλυσης. Στο κεφάλαιο 5, παρουσιάζουμε το metrical task systems problem με προβλέψεις. Στο κεφάλαιο 6, παρουσιάζουμε την μέθοδο Primal-Dual επαυξημένη με προβλέψεις. Στο κεφάλαιο 7, επανεξετάζουμε το non-clairvoyant job-scheduling problem χρησιμοποιώντας ένα διαφορετικό μοντέλο από αυτό του κεφαλαίου 3. Τέλος, στο κεφάλαιο 8, παρουσιάζουμε την συνεισφορά μας. Συγκεκριμένα, σχεδιάζουμε αλγορίθμους με προβλέψεις για το Multistage Matroid Maintenance problem.

Online αλγόριθμοι

Στο κεφάλαιο αυτό θα μελετήσουμε online αλγορίθμους. Θυμίζουμε ότι στο πλαίσιο της *competitive analysis* συγκρίνουμε την επίδοση των online αλγορίθμων με εκείνη του βέλτιστου offline αλγόριθμου, ο οποίος έχει εξ αρχής πρόσβαση σε ολόκληρη την είσοδο.

Competitive analysis

Τυπικά ορίζουμε την είσοδο ενός online προβλήματος ως μία ακολουθία από αιτήματα $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_T$. Κάθε φορά που ένας online αλγόριθμος λαμβάνει ένα αίτημα σ_t πρέπει να λάβει κάποιες αποφάσεις γνωρίζοντας μόνο τα προηγούμενα αιτήματα.

Για το υπόλοιπο αυτής της διπλωματικής θα θεωρήσουμε μόνο προβλήματα ελαχιστοποίησης. Έστω ένα πρόβλημα ελαχιστοποίησης P και έστω $\mathcal{I}$ το σύνολο των έγκυρων εισόδων για το πρόβλημα P. Συμβολίζουμε με OPT τον βέλτιστο offline αλγόριθμο για το πρόβλημα P και με $OPT(\sigma)$ το κόστος της λύσης του OPT για είσοδο $\sigma \in \mathcal{I}$.

Definition 1 (Competitive ratio)**.** *Ο αλγόριθμος ALG είναι c-competitive για ένα πρόβλημα ελαχιστοποίησης P αν για κάθε είσοδο σ υπάρχει σταθερά $a > 0$ τέτοια ώστε*

$$ALG(\sigma) \leq c \cdot OPT(\sigma) + a.$$

Λέμε ότι ο ALG έχει competitive ratio *C.*

Εναλλακτικά, μπορούμε να βλέπουμε ένα online πρόβλημα ως ένα παιχνίδι μεταξύ ενός online παίκτη και ενός κακόβουλου αντίπαλου. Ο online παίκτης τρέχει έναν online αλγόριθμο για την είσοδο που παράγει ο κακόβουλος αντίπαλος. Στόχος του παίκτη είναι να ελαχιστοποιήσει το κόστος. Αντίθετα, στόχος του κακόβουλου αντίπαλου είναι να παραγάγει την χειρότερη δυνατή είσοδο για τον παίκτη και ταυτόχρονα την ευκολότερη δυνατή είσοδο για τον βέλτιστο (offline) αλγόριθμο OPT. Μερικές φορές αναφερόμαστε στον κακόβουλο αντίπαλο και τον OPT ώς μια οντότητα, τον offline παίκτη. Για ντετερμινιστικούς αλγορίθμους ο αντίπαλος γνωρίζει ακριβώς τι θα πράξει ο online παίκτης. Για τυχαιοκρατικούς αλγορίθμους μπορούν να οριστούν διάφοροι τύποι αντίπαλου αναλόγως με την πληροφορία που αυτός έχει στην διάθεσή του [17].

- **Oblivious αντίπαλος**: Σε αυτό το μοντέλο, ο αντίπαλος γνωρίζει την περιγραφή του αλγορίθμου αλλά όχι τις τυχαίες επιλογές του και επιλύει offline το πρόβλημα.
- **Adaptive online αντίπαλος**: Σε αυτό το μοντέλο, ο αντίπαλος γνωρίζει την περιγραφή του αλγορίθμου και τις τυχαίες επιλογές του και επιλύει online το πρόβλημα.
- **Adaptive offline αντίπαλος**: Σε αυτό το μοντέλο, ο αντίπαλος γνωρίζει την περιγραφή του αλγορίθμου και τις τυχαίες επιλογές του και επιλύει offline το πρόβλημα.

Ο oblivious αντίπαλος είναι ο πιο αδύναμος ενώ ο adaptive offline αντίπαλος είναι ο ισχυρότερος. Μάλιστα ο adaptive offline αντίπαλος είναι τόσο ισχυρός που έχει αποδειχτεί ότι η τυχαιότητα δεν βοηθάει εναντίον του. Σε κάθε περίπτωση θεωρούμε το κόστος του online αλγορίθμου ως την αναμενόμενη τιμή της τυχαίας μεταβλητής που δίνει το κόστος του.

Paging

Το paging problem ένα από τα βασικότερα online προβλήματα. Για μια λεπτομερή μελέτη συνιστώνται τα κεφάλαια 3 και 4 από το [19].

Μας δίνεται ένα σύστημα μνήμης δύο επιπέδων αποτελούμενο από μία κρυφή μνήμη (μικρή και γρήγορη) και μια κυρίως μνήμη (μεγάλη και αργή). Η κρυφή μνήμη διαιρείται σε k σελίδες, ενώ η κυρίως μνήμη σε σελίδες. Όταν ζητείται μία σελίδα από τον επεξεργαστή πρέπει να φορτωθεί στην κρυφή μνήμη. Αν η σελίδα είναι ήδη στην κρυφή μνήμη έχουμε cache hit. Αν, όμως, η σελίδα δεν είναι στην κρυφή μνήμη, τότε πρέπει να διώξουμε μια σελίδα από την κρυφή μνήμη για να φορτώσουμε στην θέση της την ζητούμενη σελίδα. Στην περίπτωση αυτή λέμε ότι έχουμε cache miss ή page fault. Στόχος είναι να ελαχιστοποιήσουμε τα cache miss.

Αν ξέραμε όλη την είσοδο, τότε η βέλτιστη στρατιγική είναι να διώχνουμε την σελίδα που πρόκειται να ζητηθεί αργότερα στο μέλλον. Ο αλγόριθμος αυτός ονομάζεται κανόνας του Belady [16].

Για τους online ντετερμινιστικούς αλγορίθμους ισχύει το ακόλουθο θεώρημα.

Theorem 0.0.1. *Το competitive ratio κάθε ντετερμινιστικού αλγορίθμου για το paging problem είναι τουλάχιστον k.*

Θα ορίσουμε τώρα μια οικογένεια αλγορίθμων για το paging problem που ονομάζονται $MARKING$ αλγόριθμοι. Αρχικά, θεωρούμε ότι όλες οι σελίδες είναι σε κατάσταση unmarked. Όταν συμβαίνει ένα page fault διώχνουμε μία unmarked σελίδα και κάνουμε την ζητούμενη σελίδα mark. Αν συμβεί cache hit και πάλι κάνουμε την ζητούμενη σελίδα mark. Όταν όλες οι σελίδες είναι marked και συμβεί page fault, τότε θέτουμε όλες τις σελίδες σε κατάσταση unmarked και ξεκινάμε από την αρχή.

Theorem 0.0.2. *Ο αλγόριθμος $MARKING$ είναι k-competitive.*

Χρησιμοποιώντας τυχαιοκρατικούς αλγορίθμους μπορούμε να βελτιώσουμε το παραπάνω competitive ratio. Για τους τυχαιοκρατικούς αλγορίθμους ισχύει το ακόλουθο θεώρημα.

Theorem 0.0.3. *Το competitive ratio κάθε τυχαιοκρατικού αλγορίθμου για το paging problem είναι τουλάχιστον H_k, όπου H_k ο k-οστός αρμονικός αριθμός.*

Ορίζουμε τώρα τον αλγόριθμο $MARKER$, ο οποίος αποτελεί μια παραλλαγή του $MARKING$ αλγόριθμου. Συγκεκριμένα, ο αλγόριθμος $MARKER$ διώχνει μια unmarked σελίδα όχι βάσει κάποιου ντετερμινιστικού κριτηρίου αλλά ομοιόμορφα στην τύχη.

Theorem 0.0.4. *Ο αλγόριθμος $MARKER$ είναι $2H_k$-competitive ενάντια στον oblivious αντίπαλο.*

Metrical Task Systems (MTS)

Το Metrical Task Systems (MTS) problem προτάθηκε από τους Borodin et al. [20], σε μια προσπάθεια να συστηματοποιήσουν την μελέτη των online αλγορίθμων. Στο πρόβλημα αυτό μας δίνεται ένας μετρικός χώρος N σημείων και για κάθε αίτημα μάς δίνεται ένα διάνυσμα κόστους πάνω στα σημεία αυτά. Καλούμαστε να αποφασίσουμε σε ποιο σημείο θα ικανοποιήσουμε το αίτημα, πληρώνοντας το κόστος τού σημείου αυτού συν το κόστος για να μετακινηθούμε από το τρέχον σημείο που βρισκόμαστε στο σημείο αυτό. Στόχος είναι να ελαχιστοποιήσουμε το συνολικό κόστος. Για το MTS γνωρίζουμε ότι το βέλτιστο ντετερμινιστικό competitive ratio είναι $2-1$, ενώ για τυχαιοκρατικούς αλγορίθμους είναι $O(\log^2 N \log\log N)$ και $\Omega(\log N)$.

Συνδυασμός Online αλγορίθμων

Μια εξαιρετικά χρήσιμη τεχνική για τον σχεδιασμό online αλγορίθμων είναι ο συνδυασμός online αλγορίθμων. Συγκεκριμένα για το πρόβλημα MTS ισχύουν τα ακόλουθα θεωρήματα.

Theorem 0.0.5 (Ντετερμινιστικός συνδυασμός)**.** *Έστω m online αλγόριθμοι A_0, A_1, ..., A_{m-1} για το MTS. Υπάρχει ντετερμινιστικός αλγόριθμος με κόστος το πολύ $\frac{2\gamma^m}{\gamma-1} \cdot \min\{cost_{A_i}(I)\}$ για κάθε είσοδο .*

Theorem 0.0.6 (Τυχαιοκρατικός συνδυασμός)**.** *Έστω m online αλγόριθμοι A_0, A_1, ..., A_{m-1} για το MTS με διάμετρο D και $\epsilon < 1/2$. Υπάρχει τυχαιοκρατικός αλγόριθμος με κόστος το πολύ $(1+\epsilon)\min_i\{cost_{A_i}(I)\} + O(D/\epsilon)\ln(m)$.*

Βασικές αρχές της learning-augmented ανάλυσης

Σε αυτό το κεφάλαιο εξετάζουμε τις βασικές αρχές σχεδίασης learning-augmented online αλγορίθμων. Θεωρούμε ότι μαζί με την είσοδο του προβλήματος έχουμε πρόσβαση σε ένα μαντείο το οποίο κάνει προβλέψεις για το μέλλον. Ωστόσο, δεν έχουμε πρόσβαση στην εσωτερική λειτουργία του μαντείου ούτε ξέρουμε τον τρόπο με τον οποίο κάνει τις προβλέψεις. Για τους

αλγορίθμους που θα κατασκευάσουμε έχουμε τις εξής απαιτήσεις. Αρχικά, θέλουμε όταν οι προβλέψεις είναι ακριβείς ο αλγόριθμός μας να έχει επίδοση εφάμιλλη με αυτή του βέλτιστου offline αλγόριθμου (*robustness*), ενώ όταν οι προβλέψεις είναι ανακριβείς θέλουμε ο αλγόριθμός μας να έχει επίδοση εφάμιλλη με αυτή του βέλτιστου online αλγόριθμου (*consistency*). Για να αξιολογήσουμε την ακρίβεια των προβλέψεων ορίζουμε το συνολικό error η. Το competitive ratio $c(\eta)$ είναι, πλέον, συνάρτηση του η. Τυπικά λέμε ότι ένας online αλγόριθμος με προβλέψεις είναι γ-robust αν $c(\eta) \leq \gamma$ και β-consistent αν $c(0) \leq \beta$.

Ski rental

Θα εξετάσουμε τώρα το πρόβλημα Ski rental. Στο πρόβλημα αυτό ένας σκιέρ ξεκινάει να κάνει σκι την χειμερινή περίοδο, ωστόσο δεν γνωρίζει εκ των πρωτέρων για πόσες μέρες θα κάνει σκι. Κάθε μέρα που κάνει σκι πρέπει είτε να νοικιάσει τα πέδιλα του σκι για 1$ είτε να τα αγοράσει για b$. Ο στόχος είναι να ελαχιστοποιήσουμε τα συνολικά χρήματα που θα ξοδέψει ο σκιερ.

Έστω ότι θα κάνει σκι για T ημέρες. Η βέλτιστη λύση είναι να αγοράσει τα σκι αν $T \geq b$, αλλίως να νοικιάζει τα σκι κάθε μέρα. Φυσικά στην online εκδοχή του προβλήματος δεν γνωρίζουμε εξ αρχής τον συνολικό αριθμό ημερών που θα κάνει σκι. Στην περίπτωση αυτή, ο βέλτιστος online ντετερμινιστικός αλγόριθμος είναι να νοικιάσει τα σκι τις πρώτες $b-1$ ημέρες και να τα αγοράσει την ημέρα b. Ο αλγόριθμος αυτός είναι 2-competitive.

Έστω ότι το μαντείο προβλέψει τον αριθμό x των ημερών που θα κάνει ο σκιερ σκι. Ορίζουμε το συνολικό error ως $\eta = |y - x|$, όπου y ο πραγματικός αριθμός ημερών που θα κάνει ο σκιερ σκι.

Ο ακόλουθος αλγόριθμος έχει competitive ratio

$$1 + \min\left(\frac{1}{\lambda}, \lambda + \frac{\eta}{(1-\lambda)OPT}\right).$$

Algorithm 1

if $y \geq b$ **then**
  Buy on the start of day $\lceil \lambda b \rceil$
else
  Buy on the start of day $\lceil b/\lambda \rceil$
end if

Non-clairvoyant Job-scheduling

Θα εξετάσουμε τώρα το πρόβλημα Non-clairvoyant Job-scheduling. Στο πρόβλημα αυτό έχουμε διεργασίες και ένα μηχάνημα στο οποίο θέλουμε να τρέξουμε τις διεργασίες. Μπορούμε να διακόπτουμε την εκτέλεση μιας διεργασίας για να τρέξουμε μια άλλη διεργασία. Στην συνέχεια μπορούμε να συνεχίσουμε την πρώτη διεργασία από εκεί που σταμάτησε. Στόχος είναι να ελαχιστοποιήσουμε τον συνολικό χρόνο ολοκλήρωσης κάθε διεργασίας. Ο βέλτιστος αλγόριθμος είναι να τρέξουμε τις διεργασίες με σειρά από εκείνη με την πιο σύντομη διάρκεια στην πιο χρονοβόρα. Ο αλγόριθμος αυτός ονομάζεται Shortest-Job-First (SJF). Στην online εκδοχή δεν γνωρίζουμε τον χρόνο ολοκλήρωσης κάθε διεργασίας. Στην περίπτωση αυτή ο βέλτιστος ντετερμινιστικός αλγόριθμος είναι να τρέχουμε όλες τις διεργασίες κυκλικά και για ίσο χρονικό διάστημα. Ο αλγόριθμος αυτός ονομάζεται Round-Robin (RR).

Έστω ότι έχουμε ένα μαντείο που προβλέψει την διάρκεια κάθε διεργασίας $x_1, x_2, \ldots, x_N$. Ορίζουμε το συνολικό error ως $\eta = \sum_{i=1}^{N} |y_i - x_i|$, όπου y_i η πραγματική διάρκεια της i-οστής διεργασίας.

Συνδυάζοντας τον Round-Robin (RR) με τον αλγόριθμο Shortest-Predicted-Job-First (SPJF), ο οποίος τρέχει τις διεργασίες σε φθίνουσα σειρά με βάση την προβλεπόμενη διάρκειά τους, έχουμε έναν αλγόριθμο με competitive ratio

$$\min\{\frac{1}{\lambda}(1+\frac{2\eta}{N}), \frac{2}{1-\lambda}\},$$

όπου $\lambda \in (0,1)$.

Το πρόβλημα Paging με προβλέψεις

Στο κεφάλαιο αυτό θα εξετάσουμε το πρόβλημα Paging με προβλέψεις. Το Paging είναι το πρώτο online πρόβλημα που λύθηκε χρησιμοποιώντας προβλέψεις [51]. Στην δημοσιεύτηκε η εργασία του Rohatgi [60] η οποία βελτίωσε τα προηγούμενα αποτελέσματα. Τέλος, δημοσιεύτηκε η εργασία του Wei [66] στην οποία παρουσιάστηκε ένας βέλτιστος ντετερμινιστικός αλγόριθμος με προβλέψεις για το Paging problem.

Υποθέτουμε ότι το μαντείο προβλέπει την επόμενη χρονική στιγμή που θα ζητηθεί κάθε σελίδα $h(x_i)$. Έστω $next(x_i)$ η πραγματική επόμενη στιγμή που θα ξαναζητηθεί η σελίδα x_i. Το συνολικό error θα είναι $\eta = \sum_i |h(x_i) - next(x_i)|$.

Predictive Marker

Θα χρησιμοποιήσουμε τις προβλέψεις για να επεκτείνουμε τον αλγόριθμο $MARKER$. Συγκεκριμένα αντί να διώχνουμε μία σελίδα στην τύχη θα διώχνουμε την σελίδα που προβλέπεται να ξαναζητηθεί αργότερα στο μέλλον. Αν αντιληφθούμε ότι οι προβλέψεις είναι ανακριβείς χρησιμοποιούμε ξανά τον κλασικό $MARKER$ αλγόριθμο και διώχνουμε τις marked σελίδες ομοιόμορφα στην τύχη. Έτσι, πετυχαίνουμε competitive ratio $\min(2+4\sqrt{\frac{\eta}{OPT}}, 4H_k) = O(\min(\sqrt{\frac{\eta}{OPT}}, H_k))$.

BlindOracle

Στο [66] παρουσιάζεται μια απλούστερη προσέγγιση, η οποία έχει χρήση και σε πιο σύνθετα προβλήματα όπως θα δούμε στην συνέχεια. Συγκεκριμένα, αρκεί να συνδυάσουμε τον αλγόριθμο που ακολουθεί τυφλά τις προβλέψεις BlindOracle με τον βέλτιστο ντετερμινιστικό online αλγόριθμο για το πρόβλημα (π.χ. LRU).

Η ανάλυση για τον BlindOracle υποδεικνύει ότι έχει competitive ratio

$$\min\left(1+2\frac{\eta}{OPT}, 2+\frac{4}{k-1}\frac{\eta}{OPT}\right).$$

Συνδυάζοντας τον BlindOracle με τον LRU έχουμε έναν αλγόριθμο με competitive ratio

$$2\min\left(\min\left(1+2\frac{\eta}{OPT}, 2+\frac{4}{k-1}\frac{\eta}{OPT}\right), k\right).$$

Το πρόβλημα Metrical Task Systems με προβλέψεις

Σε αυτό το κεφάλαιο θα εξετάσουμε το πρόβλημα Metrical Task Systems με προβλέψεις [9]. Ας θυμηθούμε το προβλήμα. Μας δίνεται ένας μετρικός χώρος . Ξεκινάμε στο σημείο x_0. Κάθε χρονική στιγμή $t = 1, 2, \ldots, T$ μας δίνεται μια εργασία $\ell_t : M \to \mathbb{R} \cup \{0, +\infty\}$. Πρέπει να αποφασίσουμε αν θα μείνουμε στο σημείο x_{t-1} και θα πληρώσουμε $\ell_t(x_{t-1})$ ή θα μετακινηθούμε σε ένα νέο σημείο x_t και θα πληρώσουμε $dist(x_{t-1}, x_t) + \ell_t(x_t)$. Στόχος είναι να ελαχιστοποιήσουμε το συνολικό κόστος.

Έστω o_t το σημείο στο οποίο βρίσκεται ο optimal offline αλγόριθμος την στιγμή t. Το μαντείο θα προβλέπει την κατάσταση αυτή. Έστω p_t η πρόβλεψη για την στιγμή t. Ορίζουμε το συνολικό error ως

$$\eta = \sum_{t=1}^{T} \eta_t,\ \eta_t = dist(p_t, o_t),$$

Follow the Prediction

Ορίζουμε τον αλγόριθμο Follow the Prediction ως εξής. Έστω το σύνολο των σημείων-καταστάσεων του MTS. Κάθε χρονική στιγμή μετακινείται στο σημείο x_t:

$$x_t \leftarrow \arg\min_{x \in X} \{\ell_t(x) + 2dist(x, p_t)\}.$$

Ο αλγόριθμος Follow the Prediction πετυχαίνει competitive ratio $1 + \frac{4\eta}{OPT}$.

Συνδυάζοντας τον αλγόριθμο Follow the Prediction με έναν a-competitive ντετερμινιστικό αλγόριθμο πετυχαίνουμε competitive ratio $9 \cdot \min\left(a, 1 + \frac{4\eta}{OPT}\right)$.

Χρησιμοποιώντας τα αποτελέσματα για το MTS μπορεί κάποιος να σχεδιάσει έναν αλγόριθμο για το paging problem με competitive ratio $O(\min\{1 + \log(1 + \frac{\eta}{OFF}), \log k\})$.

Η μέθοδος Primal-Dual με προβλέψεις

Στο κεφάλαιο αυτό θα επεκτείνουμε την μέθοδο Primal-Dual για να ενσωματώσουμε προβλέψεις [12]. Η μέθοδος Primal-Dual για την κατασκευή online αλγορίθμων μελετήθηκε εκτενώς από τους Buchbinder και Naor [25].

Έστω το πρόβλημα set cover. Στο πρόβλημα αυτό έχουμε ένα σύμπαν στοιχείων $\mathcal{U}$ και ένα σύνολο συνόλων $\mathcal{S}$. Κάθε σύνολο στο $\mathcal{S}$ καλύπτει κάποια στοιχεία και έχει ένα κόστος. Στόχος μας είναι να καλύψουμε όλα τα στοιχείο με το μικρότερο δυνατό κόστος. Στην online εκδοχή του προβλήματος τα στοιχεία του σύμπαντος δεν είναι γνωστά εξ αρχής αλλά κάθε χρονική στιγμή εμφανίζεται ένα νέα στοιχείο. Το πρόβλημα set cover μπορεί να γραφτεί ως γραμμικό πρόγραμμα:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \sum_{i \in S(j)} x_i \geq 1, \forall j \\ & x_i \geq 0, \qquad \forall i \end{aligned}$$

Το δυϊκό γραμμικό πρόγραμμα είναι

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} y_j \\ \text{subject to} \quad & \sum_{j | i \in S(j)} y_j \leq c_i, \forall i \\ & y_j \geq 0, \qquad \forall j \end{aligned}$$

Έστω ότι έχουμε ένα μαντείο που προβλέπει ολόκληρη την λύση $\mathcal{A}$. Έστω $\mathcal{I}$ το στιγμιότυπο του προβλήματος που λύνουμε, $\lambda \in (0, 1)$, και $c_{ALG}(\mathcal{A}, \mathcal{I}, \lambda)$ η λύση του learning-augmented αλγορίθμου.

Algorithm 2

Whenever a new element e arrives do the following:
while $\sum_{s_i \in S(e)} x_i < 1$ **do**
  for $s_i \in S(e)$ and $s_i \in \mathcal{A}$ **do** ▷ aggressive update
    $x_i \leftarrow x_i(1 + \frac{1}{c_i}) + \frac{\lambda}{c_i|S(j)|} + \frac{1-\lambda}{c_i|S(j)\cap\mathcal{A}|}$
  end for
  for $s_i \in S(e)$ and $s_i \notin \mathcal{A}$ **do** ▷ conservative update
    $x_i \leftarrow x_i(1 + \frac{1}{c_i}) + \frac{\lambda}{c_i|S(j)|}$
  end for
end while

Ο παραπάνω αλγόριθμος έχει κόστος

$$c_{PDLA}(\mathcal{A}, \mathcal{I}, \lambda) \leq \min\{O\left(\frac{1}{1-\lambda}\right) \cdot S(\mathcal{A}, \mathcal{I}), O\left(\log\left(\frac{\lambda}{d}\right)\right) \cdot OPT(\mathcal{I})\}.$$

Η μέθοδος Primal-Dual Learning-augmented (PDLA) έχει πολλές εφαρμογές σε προβλήματα covering. Συγκεκριμένα έχει χρησιμοποιηθεί επιτυχώς για την επίλυση των ακόλουθων προβλημάτων: Ski rental problem, Bahncard problem [33], TCP acknowledgement problem [41], Parking Permit problem [54].

Το πρόβλημα Job-Scheduling με προβλέψεις

Στο κεφάλαιο αυτό επιστρέφουμε στο πρόβλημα Non-clairvoyant job-scheduling με προβλέψεις. Αυτήν την φορά όμως θα ορίσουμε ένα διαφορετικό error το οποίο θα ικανοποιεί τις εξής δύο ιδιότητες.

Property 1 (Monotonicity)**.** *For any $I \subseteq J$,*

$$ERR(\{p_j\}_{j\in J}, \{p'_j\}_{j\in J\setminus I} \cup \{p_j\}_{j\in I}) \leq ERR(\{p_j\}_{j\in J}, \{p'_j\}_{j\in J})$$

Property 2 (Lipschitzness)**.**

$$|OPT(\{p_j\}_{j\in J}) - OPT(\{p'_j\}_{j\in J})| \leq ERR(\{p_j\}_{j\in J}, \{p'_j\}_{j\in J}).$$

Η πιο σημαντική είναι η δεύτερη ιδιότητα, η οποία συσχετίζει το error με το κόστος της λύσης για το προβλεφθέν στιγμιότυπο. Συγκεκριμένα, όταν το error είναι μικρό θα πρέπει η βέλτιστη λύση για το προβλεφθέν στιγμιότυπο να είναι κοντά στην βέλτιστη λύση για το πραγματικό στιγμιότυπο.

Το απλούστερο error που ικανοποιεί τις παραπάνω ιδιότητες είναι

$$v(J, p, p') = OPT(\{p'_j\}_{j\in J_o} \cup \{p_j\}_{j\in J_u}) - OPT(\{p_j\}_{j\in J_o} \cup \{p'_j\}_{j\in J_u}),$$

όπου $J_o = \{j \in J | p'_j > p_j\}$ και $J_u = \{j \in J | p'_j < p_j\}$ είναι τα σύνολα των διεργασιών που έχουν υπερεκτιμηθεί και υποεκτιμηθεί αντίστοιχα.

Χρησιμοποιώντας το error αυτό κατασκευάζουμε τον ακόλουθο αλγόριθμο

Algorithm 3 Scheduling with predictions

$k \leftarrow 1$ and $\delta \leftarrow \frac{1}{50}$.
while $n_k \geq \frac{1}{c^3} \log n$ **do**
  $\tilde{m}_k \leftarrow Median - estimator(J_k, \delta, n)$
  $\tilde{\eta}_k \leftarrow Error - estimator(J_k, \epsilon, n, \tilde{m}_k)$
  if $\tilde{\eta}_k \geq \epsilon\delta^2 \tilde{m}_k n_k^2/16$ **then** ▷ RR (big error) round
    Process each job in J_k up to $2\tilde{m}_k$ units with Round-Robin.
  else ▷ Non-RR (small error) round
    Process jobs j with $p'_{k,j} \leq (1+\epsilon)\tilde{m}_k$ up to $p'_{k,j} + 3\epsilon\tilde{m}_k$ units in increasing order of $p'_{k,j}$.
  end if
  $k \leftarrow k+1$
end while
Run Round-Robin to complete the remaining jobs ▷ Round K+1

Ο αλγόριθμος αυτός έχει κόστος το πολύ $\min\{O(1)OPT, (1+\epsilon)OPT + 2OPT(J_{K+1}) + O(1/\epsilon^2)v\}$ με μεγάλη πιθανότητα, όπου $|J_{K+1}| \leq \frac{1}{\epsilon^3}\log n$.

Changing Bases με προβλέψεις

Στο κεφάλαιο αυτό παρουσιάζουμε την συνεισφορά μας. Συγκεκριμένα μελετάμε το Multistage Matroid Maintenance (MMM) problem στην περίπτωση που τα κόστοι είναι μοναδιαία και χρησιμοποιούμε προβλέψεις για να βελτιώσουμε τους παραδοσιακούς online αλγορίθμους. Επιπλέον, επαναξετάζουμε το paging problem και δίνουμε βέλτιστα bounds για την περίπτωση των ντετερμινιστικών αλγορίθμων.

Στο MMM πρέπει να διατηρούμε μία βάση matroid κάθε χρονική στιγμή ενόσω τα κόστη των στοιχείων αλλάζουν. Στόχος είναι να ελαχιστοποιήσουμε το συνολικό κόστος. Τυπικά το πρόβλημα ορίζεται ως εξής:

Ένα στιγμιότυπο του MMM αποτελείται από ένα matroid $\mathcal{M} = (E, \mathcal{I})$, κόστη απόκτησης $\alpha(e) \geq 0$ για κάθε $e \in E$, και για κάθε χρονική στιγμή $t \in [T]$ και στοιχείο $e \in E$, ένα κόστος κράτησης $c_t(e)$. Ο στόχος είναι η εύρεση βάσεων $\{B_t \in \mathcal{I}\}_{t\in[T]}$ που ελαχιστοποιούν το

$$\sum_t (c_t(B_t) + \alpha(B_t \setminus B_{t-1})),$$

όπου ορίζουμε $B_0 = \emptyset$

Ισοδύναμο πρόβλημα, όπως θα δούμε, είναι το *Multistage Spanning set Maintenance* (MSM) problem, όπου αντί για βάση διατηρούμε spanning set $S_t \subseteq E$ κάθε χρονική στιγμή t. Το κόστος της λύσης $\{S_t\}_{t\in[T]}$ (with $S_0 = \emptyset$) είναι

$$\sum_t (c_t(S_t) + \alpha(S_t \setminus S_{t-1})).$$

Μπορούμε να θεωρήσουμε ότι κάθε στοιχείο ζει για ένα συγκεκριμένο χρονικό διάστημα $I_e = [l_e, r_e]$ και έχει μόνο κόστος απόκτησης $a(e)$. Η παραδοχή αποδεικνύεται ότι δεν αλλάζει το πρόβλημα.

Επιπλέον, αναφέρουμε ότι υπάρχει αλγόριθμος για το MMM πρόβλημα με μοναδιαία κόστη με competitive ratio $(\log|E| \log r)$, όπου E το πλήθος των στοιχείων και r το rank του matroid.

Στην learning-augmented εκδοχή του προβλήματος έχουμε ένα μαντείο που προβλέπει τον χρόνο ζωής κάθε στοιχείου. Το error ορίζεται ως $\eta = \sum_i |t_e - h_e|$, όπου t_e και h_e ο πραγματικός και ο προβλεπόμενος χρόνος ζωής του στοιχείου e.

Ορίζουμε τον άπληστο αλγόριθμο $\mathcal{G}$ ως εξής: Αγοράζουμε το στοιχείο που θα ζήσει περισσότερο και δεν δημιουργεί κύκλο. Επαναλαμβάνουμε μέχρι να σχηματίσουμε βάση. Επιπλέον, ορίζουμε τον learning-augmented αλγόριθμο $\mathcal{B}$ (BlindOracle) ως εξής: Αγοράζουμε το στοιχείο που προβλέπεται να ζήσει περισσότερο και δεν δημιουργεί κύκλο. Και πάλι, επαναλαμβάνουμε μέχρι να σχηματίσουμε βάση.

Αποδεικνύουμε ότι

$$B \leq c\,(G + 2\eta),$$

όπου c το competitive ratio του $\mathcal{G}$.

Uniform matroids

Στα uniform matroids ο αλγόριθμος $\mathcal{G}$ είναι βέλτιστος. Δηλαδή, $c = 1$. Επομένως, το παραπάνω bound γίνεται

$$B \leq OPT + 2\eta.$$

Επιπλέον, το bound αυτό μπορεί να βελτιωθεί σε

$$B \leq OPT + \frac{4\eta}{N - r + 1},$$

όπου η είναι το συνολικό error.

Paging

Η απόδειξη για τα uniform matroids μπορεί να προσαρμοστεί για το πρόβλημα paging. Συγκεκριμένα, έχουμε ότι το κόστος του BlindOracle για το learning-augmented paging problem είναι

$$OPT + \min\left(2\eta, \frac{4\eta}{k}\right),$$

το οποίο είναι βέλτιστο για ντετερμινιστικούς αλγορίθμους.

Τέλος, σημειώνουμε ότι στην γενική περίπτωση που κάθε στοιχείο έχει διαφορετικό κόστος αποδεικνύεται ότι οι προβλέψεις που έχουμε δεν μπορούν να μας βοηθήσουν να ξεπεράσουμε την επίδοση των παραδοσιακών online αλγορίθμων [9].

Κείμενο στα αγγλικά

Chapter 1

Introduction

In a classical optimization problem we are given a specific instance of a problem and we are asked to compute an optimal solution. However, in many real-world problems the input instance is not known in advance. Instead, it is revealed gradually. Imagine for example, computing the shortest path between two cities while the traffic on the roads changes, accidents happen, roads are closed, etc. In such a problem, we have to constantly adapt to new information and make decisions without knowing the future. These problems are called *online problems* and the algorithms solving them are called *online algorithms*. Well-known online problems include the paging (caching) problem [64], the job-scheduling problem [26, 21], the portfolio selection problem [29] and many others [19]. Online problems span a wide range of domains including computer science, economics and operations research.

The following question naturally arises. How do we design, analyze and evaluate the performance of online algorithms? The traditional framework for reasoning about online algorithms is *competitive analysis*. In this framework, we compare the performance of an online algorithm to that of the optimal algorithm OPT that has knowledge of the entire input in advance.

Let us consider an online minimization problem. Suppose we design an online algorithm ALG for the problem, whose cost is at most c times that of the optimal algorithm OPT. That is, $ALG \leq c \cdot OPT$ for all instances of the problem. We say that ALG is c - competitive or that the *competitive ratio* of ALG is c.

Sometimes it is useful to think of an online problem as a game between an online player and a malicious adversary. The online player runs an online algorithm on the input generated by the adversary. The goal of the online player is to minimize the total cost incurred. On the other hand, the goal of the adversary is to generate the worst possible sequence of requests for the online player and at the same time the best sequence for the optimal offline algorithm. We sometimes refer to the adversary and the optimal offline algorithm as one entity: the offline player.

Competitive analysis has been successful in analyzing online algorithms and providing useful insight about the design of efficient online algorithms. However, it has its limitations. Namely, it is somewhat unfair and overly restrictive since it requires that an online algorithm incurs a cost of at most C times the OPT in all instances. In many real world problems there exist online algorithms that are efficient in almost all instances expect for some extremely pathological cases. Such algorithms might suffice depending on the problem. Furthermore, characterizing an online algorithm by its worst case performance causes an inability to distinguish between "good" and "bad" algorithms. Indeed, in many cases competitive analysis cannot account for the great empirical performance of some online algorithms nor differentiate between "good" and "bad" online algorithms. One such case is the paging problem, as we will become clear in the following chapters of this thesis.

In order to account for the aforementioned problems, many solutions have been proposed. Before we discuss possible solutions, we have to, first, identify what is the cause of these problems. The cause is the fact that in real-world problems the input sequence seldom is the worst possible or completely random. Indeed, most of the times the input sequence

follows predictable patterns that are exploited by empirically highly performant algorithms. This realization had led to the development of beyond-competitive analysis [46] and generally beyond-worst-case analysis [61] trying among others to reconcile the empirical performance with the theoretical analysis of online algorithms. The topics of beyond-competitive analysis and beyond worst-case analysis have recently attracted a lot of research interest. One such line of research is the field of **learning-augmented online algorithms**, which is the subject of this thesis.

As we discussed, real world instances of online problems usually obey specific patterns and thus we may design algorithms exploiting these patterns. Imagine we had an oracle predicting the future input. Such an oracle might use machine learning, heuristics or any other technique to predict patterns. How can we use these predictions to design efficient algorithms for an online problem. Before that, what should such an oracle predict? How can we measure the accuracy of the predictor? What properties should our algorithm satisfy? These are the questions we are going to answer in this thesis.

The concept of learning-augmented online algorithms was introduced by Lykouris and Vassilvtiskii in their seminal paper [51] where they used predictions to improve the classical online algorithms for the paging problem. Their work will be presented in Chapter 4 of this thesis. Recent works have been successful in developing learning-augmented algorithms outperforming traditional online algorithms for caching [60, 67], ski rental [49], scheduling [49, 56], secretary problem [10], metrical task systems [9], set cover [12], flow and matching [50], bin packing [6], and many others. Furthermore, there is a great interest in improving traditional data structures using predictions [47, 55]. Finally, a worth reading survey of learning-augmented algorithms is that of Mitzenmacher and Vassilvitskii [57].

In this thesis, we will examine many different problems and augment existing algorithms to incorporate predictions so as to take advantage of the patterns present in an instance. We will present various frameworks and provide the reader with a plethora of tools that can be employed to effectively design learning-augmented online algorithms. Until now, there is not a single "correct" framework that captures all learning-augmented online algorithms. Each one has its advantages and disadvantages as we are going to see. However, there some concepts applicable to all learning-augmented online algorithms.

To be more precise, in learning-augmented online algorithms we assume that along with the input of the problem which is revealed to us in an online fashion, we have a predictor that can predict something useful about the input. In this way, we hope to inform our decisions with the prediction and achieve better results than those of classical online algorithms. However, it might be the case that the predictions are inaccurate or even misleading and thus we should trust the predictions in a careful manner. We note that our algorithms do not have access to the inner workings of the predictor. The predictor is treated like a black-box oracle making predictions without any guarantee about its accuracy. If the predictions prove to be accurate, we hope to obtain performance comparable to the optimal offline algorithm (i.e. *consistency*). On the other hand, if the predictions are inaccurate we wish to achieve a performance similar to the optimal online algorithm (i.e. *robustness*).

We note that in general the predicted parameter is problem specific. Sometimes it seems obvious what the predictor should predict, but in many cases it not clear whatsoever. To measure the accuracy of the predictor we define the error of the predictor. The error is a function of the predicted parameter(s) and the actual instance that captures how "good" or "bad" is the predictor. The definition of the error is also problem specific. Furthermore, recent work [37] suggests that properties other than consistency and robustness might also be desirable, as we will see in Chapter 7 of this thesis.

We will now present a method that can guide us in order to find what to predict and define the error of the predictor. First, we have to ask what the optimal algorithm does. In many cases the optimal algorithm is a greedy algorithm making decisions based on some

parameters of the problem. If this is the case, then these parameters are a good candidate for a prediction. Moreover, if the optimal algorithm is not a simple greedy algorithm there might exist a greedy algorithm that is a good approximation of the optimal algorithm. In this case, we can again identify what to predict using the greedy algorithm and hope for a good approximation of the optimal algorithm. The error of the prediction in this setting is usually defined as the ℓ_1-norm between the true and predicted parameters of the problem.

In many problems, however, a simple greedy algorithm that makes decisions based on some parameters of the problem does not exist. Instead the optimal algorithm makes decisions based on the entire input. In such a case, we can employ an alternative approach. The online algorithm can be seen as being in a configuration or a state at each timestep. The definition of the state is problem specific, nonetheless it applies to almost all online problems. In these problems, the predictor may predict the states of the optimal algorithm in each timestep. The error is defined as the sum of the costs (or the distance) of moving from the predicted state p_t, at timestep t, to the respective state of the optimal algorithm o_t. We will examine this approach in chapter 5 of this thesis.

Finally, if none of the above works, one might employ the following approach. The predictor will predict the entire solution and the error will be the difference between the predicted and the optimal solution. We will examine this approach in chapter 6 of this thesis.

To sum up we have the following categories:

- If there exists a optimal greedy algorithm for our problem which makes decisions based on a parameter p_t of the problem, we try to mimic the greedy algorithm and the oracle predicts p_t at time t. The error is defined as $\eta = \sum_t |p_t - p'_t|$, where p'_t is the predicted value of p_t.

- If the optimal algorithm makes decisions based on the entire input, the oracle predicts the states of the optimal algorithm. The error is defined as $\eta = \sum_t cost(p_t, o_t)$, where o_t is the state of the optimal algorithm at time t, p_t the predicted state and $cost(p_t, o_t)$ the cost of moving from the predicted to the true state of the optimal algorithm

- The oracle predicts the entire solution. The error is defined as $\eta = S - OPT$, where S is the cost of the predicted solution and OPT is the cost of the optimal solution.

The next issue we have to address is how to design an algorithm that takes into consideration the prediction and is robust and consistent. Again, there is not a single approach that works in every case, however, there seems to be a common pattern applicable to almost all settings. The idea is to combine the algorithm following the predictor (either blindly or with some caution) with an optimal online algorithm for the problem of interest. First, we have to bound the competitive ratio of the algorithm following the predictor using the total error of the predictor η. Then, as we will see in chapter 2, it is possible to combine multiple online algorithms into one algorithm that performs close to the best of them. Thus, we combine the algorithm following the predictor with the optimal online algorithm. In this way, if the predictions are accurate, we have an algorithm close to the optimal offline algorithm (consistency). On the other hand, if the predictions are inaccurate we fall back to the optimal online algorithm (robustness).

1.1 Contribution

In this thesis, apart from a presentation of the most fundamental results in the domain of learning-augmented online algorithms, we will study the problem of *Multistage Matroid Maintenance (MMM)* problem with predictions. This problem (without predictions) was studied by Gupta, Talwar, Wieder in [35] .

In most combinatorial optimization problems one is concerned with solving an instance frozen in time and usually the goal is to minimize the total cost incurred. However, in real-world problems one has to solve an instance of the same problem repeatedly. Moreover, the underlying costs might change over time. One approach is to resolve the new instance from scratch, however, in most cases there is a transition cost between the old and the new solution. Thus, it might be better to start with the old solution and carefully transform it to a solution for the new instance.

Imagine, for example, having to solve the minimum spanning tree problem in a graph where at each timestep some edges died and some new edges were born. Finding the minimum spanning tree at each timestep separately may lead to more costly solutions. It might be better to retain the spanning tree of the previous timestep and add the necessary edges to complete the new spanning tree. It is thus a game of balance between acquiring new edges and retaining the edges of the old spanning tree.

Formally, we will examine the *Multistage Matroid Maintenance problem (MMM)*, where the underlying structure is a matroid and we have to maintain a base. We will formally define what a base of a matroid is in the last chapter. For now, it suffices to think of a matroid base as a spanning tree of a graph. We encourage the reader to think in terms of spanning trees of graphs (i.e. graphical matroids) in order to intuitively grasp the ideas we are going to present. In our analysis, we will concentrate on the case where all the costs are uniform.

In the learning-augmented setting, we have an oracle predicting the time each edge will die. The error is defined as the ℓ_1 norm between the true and the predicted death times of the edges. This formulation is analogous to that of the paging problem with predictions that we will examine in chapter 4 of this thesis.

Using the predictions we will try to mimic the greedy algorithm G buying (when needed) the edge dying furthest in the future. The competitive ratio of algorithm G is conjectured to be constant (in the case of uniform matroids). The known bound is $O(\log T)$ where T is the entire time horizon. Furthermore, a trivial bound of r, where r is the rank of the underlying matroid also holds. Note that depending on the specific matroid of interest the competitive ratio of G might be further improved. For example, in the case of uniform matroids algorithm G is optimal. In our results, we assume that the competitive ratio of G is c.

We will design a learning-augmented algorithm for the Multistage Matroid Maintenance problem that is both robust and consistent.

Theorem 1.1.1. *Let A be an a-competitive online algorithm for the Multistage Matroid Maintenance problem. There exists an online algorithm for the Multistage Matroid Maintenance problem with predictions that achieves a competitive ratio of at most*

$$O\left(\min\left(a, c^2 + 2c\frac{\eta}{OPT}\right)\right),$$

where η is the total prediction error, OPT the optimal offline algorithm and c the competitive ratio of G.

In uniform matroids algorithm G is optimal and the above result is tight up to constant factors.

Finally, we show that our analysis is also applicable in the paging problem with predictions. We will obtain previously known optimal results in a simpler and more intuitive manner.

1.2 Organization

The structure of this thesis is as follows. In chapter 2 we define online algorithms and competitive analysis which is the main tool used to analyze online algorithms. We also examine two fundamental online problems - the paging problem and the metrical task systems

problem. Chapters 3-7 are a presentation of the most significant results in the domain of learning-augmented online algorithms. In chapter 3, we introduce learning-augmented online algorithms by studying the ski rental problem and the non-clairvoyant job-scheduling problem. In chapter 4, we examine the paging problem with predictions problem, which is the problem that initiated the domain of learning-augmented online algorithms. In chapter 5, we examine the metrical task systems with predictions problem. In chapter 6, we present the learning-augmented primal-dual method for linear programs. In chapter 7, we revisit the non-clairvoyant job-scheduling problem we presented in the third chapter. This time, however, we consider a different framework. Finally, in chapter 8 we present our novel results. Namely, we design learning-augmented algorithms for the Multistage Matroid Maintenance problem.

Chapter 2

Online Algorithms

In traditional optimization problems an algorithm is given an instance of a problem and has to output an optimal solution usually minimizing a cost function or maximizing a profit function. In online computation, however, the instance of the problem is not known up-front. Instead, it is given to the algorithm gradually in a piecewise fashion. The algorithm has to make decisions based on past events and without information about the future. Imagine, for example, having to manage the assets of a company. In which products should we invest and what is the optimum strategy for generating revenue. The future prices of the market are not known in advance nor is the demand for the company's products. Thus, we have to make decisions based on incomplete information.

Many real-world problems are intrinsically online problems and thus the study of online algorithms is of a great significance. Well-known online problems include the paging (caching) problem [64], the job-scheduling problem [26, 21], the portfolio selection problem [29] and many others [19]. Online problems span a wide range of domains including computer science, economics and operations research.

In this section, we will examine the framework of competitive analysis, which is the standard framework used to study online algorithms. In this framework, we model online problems as a game between an online player and a malicious adversary. The online player tries to minimize the overall cost, while the adversary constructs the worst-case input for the online player. We compare the performance of the online player to that of the optimal offline algorithm.

Note that while this framework has succeeded in modeling online problems and providing many useful insights, it has been criticised as being overly pessimistic. Indeed, in many cases competitive analysis cannot account for the great empirical performance of some online algorithms nor differentiate between "good" and "bad" online algorithms.

For example, in the paging problem, the discrepancy between empirical performance and theory is evident. The LRU algorithm outperforms almost all other algorithms, but under the lens of competitive analysis most of them have the same performance.

This is due to the fact that in real-world problems the input sequence seldom is the worst possible. Indeed, the input sequence follows predictable patterns that are exploited by algorithms such as the LRU. This realization has led to the development of beyond-competitive- analysis [46] and beyond-worst-case analysis [61]. Lately, the topics of beyond-competitive analysis and beyond worst-case analysis have attracted a lot of research interest. One such line of research is the field of learning-augmented online algorithms, which is the concern of this thesis.

We will now present the competitive analysis framework. In the next chapter, will introduce the learning-augmented framework for online algorithms.

2.1 Competitive Analysis

The standard framework used to analyze and evaluate the performance of online algorithms is *competitive analysis*, which was introduced by Sleator and Tarjan [64]. The performance of an

online algorithm is compared to that of the optimal offline algorithm which has knowledge of the entire input sequence in advance. The comparison is done using the notion of competitive ratio.

Formally, the input of an online problem is a sequence of requests $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_T$. When a request is received our algorithm has to perform some actions to serve that request. Note that our algorithm has access only to previous requests and not to future requests. An algorithm that solves an online problem is called an *online algorithm*. On the other hand, an algorithm that receives the entire input in advance is called an *offline algorithm*.

For the rest of this thesis we will only consider minimization problems. The definitions for maximization problems are analogous. Let $\mathcal{P}$ be a minimization problem and $\mathcal{I}$ be the set of all valid inputs of the problem. We denote by OPT the optimal offline algorithm for problem P and by $OPT(\sigma)$ the cost of the solution produced by OPT on an input sequence $\sigma \in \mathcal{I}$.

Definition 2. *(Competitive ratio) We say that an online algorithm ALG for a minimization problem P is c-competitive if for every input σ there exists a constant $a > 0$ such that*

$$ALG(\sigma) \leq c\, OPT(\sigma) + a.$$

We say that algorithm ALG attains a competitive ratio *c.*

If constant a is zero, we might say, for emphasis, that algorithm ALG is *strictly c-competitive*.

Sometimes it is useful to think of an online problem as a game between an online player and a malicious adversary. The online player runs an online algorithm on the input generated by the adversary. The goal of the online player is to minimize the total cost incurred. On the other hand, the goal of the adversary is to generate the worst possible sequence of requests for the online player and at the same time the best sequence for the optimal offline algorithm. We sometimes refer to the adversary and the optimal offline algorithm as one entity: the offline player. For deterministic algorithms the adversary knows in advance what will be the response of the online algorithm to every request. For randomized algorithms, however, we define the following types of adversaries [17].

- Oblivious adversary: The adversary knows the online algorithm, but not its random choices. Moreover, it has to construct the entire input sequence in advance. In this case, the competitive ratio c of a randomized algorithm ALG is defined as

  $$\mathbb{E}[ALG(\sigma)] \leq c\, OPT(\sigma) + a,$$

  where the expectation is taken over all possible realizations of the randomized algorithm ALG.

- Adaptive-online adversary: The adversary knows the online algorithm and its random choices. However, the adversary must also serve the requests online. In this case, the competitive ratio C of a randomized algorithm ALG is defined as

  $$\mathbb{E}[ALG(\sigma)] \leq c\, ADV(\sigma) + a,$$

  where $ADV(\sigma)$ is the cost of the online adaptive algorithm.

- Adaptive-offline adversary: The adversary knows the online algorithm and its random choices. Moreover, it serves the requests offline. In this case the competitive ratio C of a randomized algorithm ALG is defines as

  $$\mathbb{E}[ALG(\sigma)] \leq c\, OPT(\sigma) + a$$

The oblivious adversary is the "weakest" adversary, while the adaptive-offline adversary is the "strongest" adversary. It has been shown that randomization does not give an advantage against an adaptive-offline adversary.

2.2 Paging

We are now going to examine the paging (caching) problem, which is a canonical example of an online problem and has been extensively studied. We will present the basic results of paging since it is a fairly simple problem illustrating the fundamental techniques used in competitive analysis. For a detailed analysis we refer the reader to chapters 3 and 4 of [19].

We are given a two-level memory system comprised of a cache (small and fast) memory and a main (large and slow) memory. The cache is divided into k sections called pages, while the main memory contains N pages. When a page is requested we have to load it to the cache in order to be accessed by the CPU. If the page is already in the cache, we have a cache hit. If the page is not in the cache, we have to evict a page and load the requested page. In this case, we have a cache miss or a page fault. The goal is to devise an eviction policy such that the number of page faults is minimized.

If we knew the entire request sequence in advance, the optimal strategy would be to evict the page that will be requested again furthest in the future. This policy is known as Belady's rule [16].

2.2.1 Deterministic paging algorithms

There exist many deterministic algorithms for the paging problem. The LRU (Least Recently Used) algorithm, which evicts the page whose most recent request was earliest, is the most popular and widely used in practice. Another policy is the FIFO (First in First out) algorithm which evicts the page that has been in the cache the longest.

We will now prove that every deterministic algorithm is k-competitive.

Theorem 2.2.1. *The competitive ratio of any deterministic algorithm ALG for the paging problem is at least k.*

Proof. Consider an instance of caching where there exist $k+1$ pages in total. In the beginning, the cache contains pages $p_1, p_2, \ldots, p_k$. The adversary makes the following request sequence. First, it requests page p_{k+1} and ALG incurs a cache miss. In order to serve the request ALG has to evict a page p_i from the cache. Then the adversary will request this page and ALG will again suffer a cache miss. The adversary will continue this strategy for T requests. Thus, ALG will pay a total cost of T. We claim that the optimal algorithm will pay at most $\frac{T}{k}$. The optimal algorithm OPT evicts the page that will be requested furthest in the future. Thus, after a page fault OPT is guaranteed to serve the next k-1 requests without suffering a page fault. Therefore, the total cost of OPT is at most $\frac{T}{k}$. □

We will now define a family of algorithms called the $MARKING$ algorithm. Initially, all the pages of the cache are in the unmarked state. When a page fault happens the $MARKING$ algorithm evicts an unmatched page and marks the requested page. If it is a cache hit, the algorithm marks the requested page (if it is unmarked). When all the pages are marked and a page fault happens the algorithm unmarks all the pages and starts from the beginning.

Depending on which unmarked page the $MARKING$ algorithm evicts we have different algorithms. Algorithm LRU is an example of a MARKING algorithm. We state the following theorem regarding the competitive ratio of the MARKING algorithm.

Theorem 2.2.2. *The $MARKING$ algorithm is k-competitive.*

2.2.2 Randomized paging algorithms

Using randomization one can achieve a better competitive ratio. In this section we will assume an oblivious adversary.

First, we state the following lower bound about any randomized online algorithm for the paging problem.

Theorem 2.2.3. *The competitive ratio of any randomized online algorithm for the paging problem is at least H_k, where H_k is the k-th harmonic number.*

We now define the $MARKER$ algorithm, which is a modification of the $MARKING$ algorithm we presented in the previous section. When a page fault happens $MARKER$ a evicts an unmarked page not based in a deterministic criterion but uniformly at random. We state the following theorem regarding the competitive ratio of the MARKER algorithm.

Theorem 2.2.4. *The $MARKER$ algorithm is $2H_k$-competitive against any oblivious adversary.*

2.3 Metrical Task Systems

In this section we are going to present the Metrical Task Systems (MTS) problem, which was proposed by Borodin, Linial and Sacks. [20]. The MTS problem captures many online problems including paging, list accessing and k-server. It serves as a unifying framework for the analysis of online algorithms. In the early stages of competitive analysis most problems were analyzed in a rather ad-hoc way. With the introduction of the MTS problem a more rigorous approach was possible.

First, we have to define the notion of a *metric space*

Definition 3. *A metric space is defined as a pair (S, d) where S is a set of points and $d : S \times S \to \mathbb{R}$ that satisfies:*

- $d(i, j) > 0, \forall i \neq j, \quad i, j \in S$
- $d(i, i) = 0, \forall i \in S$
- $d(i, j) + d(j, k) \geq d(i, k), \forall i, j, k \in S$
- $d(i, j) = d(j, i), \forall i, j \in S$

We may think of the points as states or different "configurations" the online player can be situated in. The metric d is the transition cost between these configurations.

In order to model online algorithms we have to define the input sequence. At each timestep a *task* arrives and has to be processed by the online player. A task is an N-ary vector of costs $r = (r(1), r(2), ...r(N))$ where each $r(i)$ is the cost of processing the task while in state i. The online player may choose to move from a state i to a different state j before processing the task and pay $d(i, j) + r(j)$. The objective is to minimize the total cost.

The optimal competitive ratio for deterministic algorithms is $2N - 1$. For randomized algorithms the optimal competitive ratio is $O(\log^2 N \log \log N)$ and $\Omega(\log N)$ [20].

Usually, we study special cases of the MTS problem where the metric space has some specific structure. For a detailed analysis of the MTS problem we refer the reader to [19]. In this thesis, apart from the definition of the problem no other knowledge is required.

We will now model the paging problem, we examined in the previous section, as an MTS problem. The states of the MTS will be all the different k-subsets of N pages. In order to move from one cache configuration s_1 to another another s_2 we have to evict elements of s_1 that are not in s_2 and load the respective elements of s_2. That is, the distance of two states is the number of different elements between them. Formally, $d(s_i, s_j) = k - |s_i \cap s_j|$. It is easy to see that d is indeed a metric. We now have to define the tasks. When a page request happens we have to include the requested page, say p_i in the cache. Thus, we define the cost vector as

$$r_i(s_j) = \begin{cases} 0, & p_i \in s_j \\ \infty, & p_i \notin s_j \end{cases}$$

2.4 Combining online algorithms

A very powerful technique that will be used extensively in this thesis is the combination of online algorithms. That is, if we have many online algorithms for an online problem we can construct an online algorithm that is approximately as good as the best of them. We will state two theorems regarding the deterministic [32] and randomized [18] combination of online algorithms for the metrical task systems problem. We will present them with respect to the MTS problem - see section 2 in [9], but they can be used in other settings as well.

Theorem 2.4.1 (Deterministic combination)**.** *Given m online algorithms A_0, ..., A_{m-1} for an MTS, there exists a deterministic algorithm achieving cost at most $\frac{2\gamma^m}{\gamma-1} \cdot \min_i\{cost_{A_i}(I)\}$, for any input sequence I.*

The optimal choice for γ is $\frac{m}{m-1}$. Then $\frac{2\gamma^m}{\gamma-1}$ becomes 9 for $m = 2$.

Algorithm 4 MIN

```
choose 1 < γ ≤ 2
ℓ = 0
repeat
    i = ℓ mod m
    while cost(A_i) ≤ γ^ℓ do
        follow A_i
    end while
    ℓ = ℓ + 1
until the end of the input
```

Algorithm 4 works in a cyclical pattern. It follows an algorithm for some time and then changes to the next algorithm. When all algorithms have been used, MIN switches back to the first algorithm completing the circle. The algorithm takes advantage of the fact that the cost of switching between A_i and A_{i+1} can be bounded by the cost so far of A_i plus the cost so far of A_{i+1}. For the proof we refer the reader to Appendix A of [9]. Next, we state a theorem regarding the randomized combination of online algorithms.

Theorem 2.4.2 (Randomized combination)**.** *Given m online algorithms A_0, ..., A_{m-1} for an MTS with diameter D and $\epsilon < 1/2$, there is a randomized algorithm, such that for any instance I, its expected cost is at most*

$$(1+\epsilon)\ \min_i\{cost(A_i(I))\} + O(D/\epsilon)\ln(m).$$

Chapter 3

Basics of Learning-augmented Analysis

In this chapter we will examine two well-studied online problems, Ski rental and Non-clairvoyant job scheduling, as well as their respective learning-augmented versions. Through the analysis of these problems we will introduce the basic concepts and key-ideas of learning-augmented online algorithmic analysis. This section is based on the results of Kumar, Purohit and Svitkina [49] which was one of the first papers in the field, along with [51] which will be examined in the following chapter.

In all learning-augmented online algorithms we assume that along with the input of the problem which is revealed to us in an online fashion, we are given a predictor that can predict something useful about the input. Using the prediction, we hope to inform our decisions and achieve better results than those of classical online algorithms. However, it might be the case that the predictions are inaccurate or even misleading and thus we should trust the predictions in a careful manner. We note that our algorithms do not have access to the inner workings of the predictor. The predictor is treated like a black-box oracle without any guarantee about its accuracy. If the predictions prove to be accurate, we want to obtain performance comparable to the optimal offline algorithm (i.e. *consistency*). On the other hand, if the predictions are inaccurate we wish to achieve a performance similar to the optimal online algorithm (i.e. *robustness*).

We note that in general the predicted parameter is problem specific. Many times it seems obvious what the predictor should predict, but in many cases it not clear whatsoever. To measure the accuracy of the predictor we define the error of the predictor. The error is a function of the predicted parameter(s) and the actual instance that captures how "good" or "bad" is the predictor. The definition of the error is also problem specific.

Moreover, issues of practicality as well as learnability naturally arise since it might be the case that a "natural" predictor, that intuitively makes sense, is not learnable or that an "unnatural" and maybe impractical predictor is indeed learnable. We, again, emphasize that the choice of the predictor is ad-hoc and we do not have a formal model classifying the different types of predictors along with their respective errors.

Summing up, an online algorithm using predictions should satisfy the following desiderata. First, the predictor should be treated as a *black-box oracle*. That is it has no access to the inner workings of the predictor and there is no guarantee about the accuracy of the predictor. Second, the algorithm should be *consistent*. That is, if the predictions are accurate (i.e. the error is zero) the performance of the algorithm should be close to that of the best offline optimal algorithm. Third, the algorithm should be *robust*. That is, even if the predictions are inaccurate, the performance of the algorithm should be close to that of the optimal online algorithm without predictions.

Finally, in order to evaluate the performance of a learning-augmented online algorithm, we use the notion of the competitive ratio as it was discussed in chapter 2 of this thesis. However, the competitive ratio c of a learning-augmented algorithm is a function of the predictor h and the error and thus we write $c = c(h, \eta)$.

We will now formally define all of the above. Let η be the error of the predictor h and $c(h, \eta)$ be the competitive ratio of a learning-augmented algorithm using the predictor h. We

usually ommit the predictor h and write the competitive ratio only as a function of η - that is, $c = c(\eta)$. We say that the algorithm is γ-robust if $c(\eta) \leq \gamma$. Moreover, we say that the algorithm is β-consistent if $c(0) \leq b$.

3.1 Ski rental

In this section we introduce the well-known ski rental problem, which is a paradigm problem in the class of rent/buy problems. In these problems, one has to either pay a small repeating fee or pay a one-time large amount of money eliminating the fee. Imagine a company that needs certain computing resources so as to support its clients. Every day a potentially different amount of customers need to be serviced and thus the demand in computing resources might vary. The company may rent machines on the cloud for a small fee, but for a limited amount of time, or otherwise buy its own machines for a large amount of money. Each day there is a specific demand for servers and the company must decide whether to rent or buy machines so as to handle the demand. This is a generalization of the Ski rental problem (see [44]), but the results we are going to present for Ski rental can easily be applied to this more general setting. Another extension of the Ski rental problem is the Parking permit problem [54]. We will examine the Parking permit problem in chapter 6 of this thesis. Rent or buy problems are very common in practice and thus the design of efficient algorithms is of great significance.

3.1.1 The problem

Imagine a skier facing the following dilemma. At the beginning of winter, the ski center opens and the skier skis everyday. Of course, he should either rent the skis or buy them. He may rent skis for \$1 per day or buy the skis for \$b. However, the skier does not know when the ski center will close and thus when he will have to stop skiing. The goal is to minimize the total money spent. What is the optimal strategy for the skier?

Suppose that the skier is going to ski for T days and then stop. The optimal offline algorithm OPT buys the skis if $T \geq b$. Otherwise, OPT rents skis each day. Thus $OPT = min(T, b)$

In the online setting, the skier does not know in advance the number of skiing days T. Each day, he has to decide whether to rent or buy skis. The optimal deterministic strategy is the break-even algorithm. Rent skis for the first b-1 days and buy them on day b. If $T < b$, then our algorithm pays the same as the optimal offline algorithm. If $T \geq b$, then our algorithm will rent for the first b-1 days and buy skis on day b paying a total of 2b-1. The optimal offline algorithm buys the skis on the first day paying a total of b. Thus, our algorithm is 2-competitive. One can prove that there is no better deterministic algorithm. There exists, however, a randomized algorithm that yields a competitive ratio of $\frac{e}{e-1}$ [43].

3.1.2 Ski rental with predictions

The essence of the problem lies in the uncertainty of the skier about the future. Namely, the skier does not know the number of skiing days. In practice, however, the skier can make a reasonable guess, for example, based on weather forecasts or past behavior of other skiers. Suppose that he is given access to a black-box oracle predicting the number of skiing days T. Is it now possible to achieve a better competitive ratio than the break-even algorithm?

Formally, let x be the actual number of skiing days and y be the predicted number of skiing days. We define the error of the predictor as $\eta = |y - x|$. Let OPT denote the optimal offline algorithm, B the algorithm following the predictor and A the break-even algorithm discussed in the previous section. We sometimes abuse the notation and denote by OPT, B, A the respective costs of the algorithms.

First, we note that B, although consistent, is not robust. That is, if $\eta = 0$, the algorithm pays the same as OPT, but if $\eta > 0$ the cost of B is unbounded. For example, if $y > T > b$ then B pays T, while OPT just pays b.

We now present the algorithm of Kumar et al. [49] that is both consistent and robust.

Let $\lambda \in (0, 1)$ be a tunable hyper-parameter capturing our trust in the predictor. As $\lambda \to 0$, our trust in the prediction is increased.

Algorithm 5

if $y \geq b$ **then**
  Buy on the start of day $\lceil \lambda b \rceil$
else
  Buy on the start of day $\lceil b/\lambda \rceil$
end if

A simple case analysis shows that the competitive ratio of Algorithm 5 is bounded by

$$1 + min\left(\frac{1}{\lambda}, \lambda + \frac{\eta}{(1-\lambda)OPT}\right).$$

Thus, *Algorithm 1* is $(1 + \frac{1}{\lambda})$-robust and $(1 + \lambda)$-consistent.

What *Algorithm 1* actually does is the following. It modifies algorithm A to cautiously take into account the prediction of the oracle. If the oracle predicts that the skier will ski for more than b days, *Algorithm 1* buys the skis a bit earlier than A. On the other hand, if the oracle predicts that the skier will ski for less than b days, *Algorithm 1* waits a bit longer than A. Thus, *Algorithm 1* moves the buying threshold of A from b to $\lceil \lambda b \rceil$ or $\lceil b/\lambda \rceil$ respectively depending on the prediction.

Another equivalent way to interpret the above algorithm is as follows. We simulate the execution of algorithms A and B (in parallel). Each day we choose to follow one of them. In this way, we effectively combine the two algorithms. This interpretation will reveal a key idea commonly used in the analysis of online algorithms. Namely, we can combine online algorithms as long as we can simulate their execution and bound the cost for switching between them. For a more detailed analysis see [32], [18]. We give an equivalent algorithm to *Algorithm 1* that demonstrates the above idea. We note that switching from an algorithm ALG_1 to another algorithm ALG_2 in the problem of Ski rental simply means that if ALG_2 has already bought skis in the past, the combined algorithm should also buy the skis.

Algorithm 6

if $y \geq b$ **then** ▷ B buys skis on the first day.
  Follow algorithm A.
  Switch to B when $A \geq \lceil \lambda b \rceil$.
else ▷ B rents each day.
  Follow algorithm B.
  Switch to A when $B \geq \lceil \frac{\lambda}{b} \rceil$.
end if

It is easy to see that Algorithm 6 is actually identical to Algorithm 5. Thus, Algorithm 6 is also $(1 + \frac{1}{\lambda})$-robust and $(1 + \lambda)$-consistent.

Except for deterministic algorithms one may also design randomized algorithms using the prediction. In [49], they present a randomized algorithm achieving a competitive ratio of at most $min\{\frac{1}{1-e^{-(\lambda-\frac{1}{b})}}, \frac{\lambda}{1-e^{-\lambda}}(1 + \frac{\eta}{OPT})\}$

Another model for learning-augmented Ski rental will be presented in chapter 6 where we introduce the primal-dual method [12]. We should also note [45] where they use an oracle

providing an estimate of the probability that there will be less than a threshold number of ski-days, and [5] where they consider a different model.

3.1.3 Robustness-consistency trade-off

An important aspect of Algorithm 5 is that it implies a trade-off between robustness and consistency. Setting λ close to zero yields a better competitive ratio when η is small. On the other hand, setting λ closer to one yields a more robust algorithm when the predictions are inaccurate.

The question of whether such trade-offs are necessary arises naturally. And if so, what are the optimal trade-offs? This question is answered in [67]. Specifically, they prove the following theorems:

Theorem 3.1.1. *Let $\lambda \in (0,1)$ be a fixed parameter. Any $(1+\lambda)$-consistent deterministic algorithm for ski-rental with predictions is at least $1+\frac{1}{\lambda}$-robust.*

Theorem 3.1.2. *Any (randomized) algorithm for ski-rental with predictions that achieves robustness γ must have consistency*

$$\beta \geq \gamma log(1+\frac{1}{\gamma-1}).$$

3.2 Non-clairvoyant job scheduling

In this section we will use predictions to solve the Non-clairvoyant job scheduling problem. The Non-clairvoyant job scheduling problem has been extensively studied and there exist numerous extensions of this problem ([14], [15]). We will examine the most basic variation of the problem and provide learning-augmented algorithms to achieve better performance than the classical online algorithms.

3.2.1 The problem

Suppose we have a single machine on which to run n jobs with unknown completion times. A job can be preempted at any time and resumed later. The goal is to minimize the sum of completion times of the jobs. The optimal offline algorithm is to schedule jobs in non-decreasing order of job lengths, i.e. shortest job first (SJF). The optimal deterministic offline algorithm is running jobs in equal portions and in circular order. This algorithm is called round-robin and achieves a competitive ratio of 2 [58].

3.2.2 Non-clairvoyant job scheduling with predictions

Suppose that $x_1, x_2, ..., x_n$ are the actual processing times of the n jobs and we have an oracle predicting these processing times. Let $y_1, y_2, ..., y_n$ be the predicted lengths of the jobs. We define the prediction error for the i-th job to be $\eta_i = |y_i - x_i|$. The total error of the predictor is $\eta = \sum_{i=1}^{n} \eta_i$.

To incorporate the predictions we will combine round-robin with the algorithm following blindly the prediction. In [49] they name this algorithm Shortest Predicted Job First(SPJF). Before we present how to combine the two algorithms, we define the notion of monotonic algorithms.

We call a non-clairvoyant scheduling algorithm monotonic if it satisfies the following property: given two instances with job lengths $(x_1, x_2, ..., x_n)$ and $(x_1^{'}, x_2^{'}, ..., x_n^{'})$ such that $x_i \leq x_i^{'}$, for all i, the sum of completion times for the first instance is less than the second one. We note that both RR and SPJF are monotonic.

We now present the lemma allowing us to combine algorithms for the non-clairvoyant job scheduling problem.

Lemma 3.2.1. *Given two monotonic algorithms with competitive ratios α and β for the minimum total completion time problem with preemptions, and a parameter $\lambda \in (0,1)$ one can obtain an algorithm with competitive ratio $min\{\frac{\alpha}{\lambda}, \frac{\beta}{1-\lambda}\}$.*

Proof. Let A and B be the two algorithms. The combined algorithm runs the two given algorithms in parallel. Algorithm A is run at a rate of λ and B at a rate of $1-\lambda$. Thus, A becomes $\frac{\alpha}{\lambda}$-competitive since all times increase by a factor of $\frac{1}{\lambda}$. Similarly,B becomes $\frac{\beta}{1-\lambda}$-competitive. We note that the fact that some jobs are concurrently being executed by both A and B only decreases the above competitive ratios. Thus, the combined algorithm has a competitive ratio of $min\{\frac{\alpha}{\lambda}, \frac{\beta}{1-\lambda}\}$. □

Next, we bound the performance of SPJF with respect to η.

Lemma 3.2.2. *The SPJF algorithm has competitive ratio at most $1 + \frac{2\eta}{n}$.*

Proof sketch. For simplicity assume that $x_1 \leq x_2 \leq ... \leq x_n$. Let ALG denote the sum of completion times of jobs when running SPJF. We define $d(i,j)$ as the amount (in time units) of job i that has been completed before job j. We have that

$$ALG = \sum_{i=1}^{n} x_i + \sum_{(i,j):i<j} (d(i,j) + d(j,i)).$$

If $i < j$ and $y_i < y_j$, then the shorter job is scheduled first and $d(i,j) + d(j,i) = x_i + 0$. If, however, $i < j$ and $y_i \geq y_j$, then the longer job is scheduled first and $d(i,j) + d(j,i) = 0 + x_j$. Using these observations one can prove that

$$ALG \leq OPT + (n-1)\eta \Rightarrow \frac{ALG}{OPT} \leq 1 + \frac{(n-1)\eta}{OPT}$$

□

Finally, using the fact that all jobs have length at least 1 and thus $OPT \geq \frac{n(n+1)}{2}$, we have that

$$ALG \leq OPT + \frac{2\eta}{n}.$$

Using Lemma 3.2.1 and Lemma 3.2.2, as well as the fact that RR is 2-competitive, we have the following theorem.

Theorem 3.2.3. *The preferential round-robin algorithm with parameter $\lambda \in (0,1)$ has competitive ratio at most $min\{\frac{1}{\lambda}(1 + \frac{2\eta}{n}), \frac{2}{1-\lambda}\}$. In particular, it is $\frac{2}{1-\lambda}$-robust and $\frac{1}{\lambda}$-consistent.*

We will return to this problem in chapter 7 viewing it from a different perspective. We also note that there exists the work of Mitzenmacher [56] where an alternative model is considered.

3.2.3 Robustness-consistency trade-off

As in Ski rental, we observe that there exists a trade-off between robustness and consistency. That is, when $\lambda \to 0$ (we have confidence in the predictor) consistency is close to 2, while robustness diverges to infinity. On the other hand, when $\lambda \to 1$ (we do not trust the predictor) consistency diverges to infinity, while robustness is bounded.

In [67] they prove the following lower bound: Any $(1+\lambda)$-consistent algorithm for non-clairvoyant scheduling with predictions must have robustness

$$\gamma \geq \frac{n + n(n+1)\lambda}{1 + \lambda(n+1)(n+2)/2}.$$

This bound is tight when $\lambda = 0$ and when $\lambda = 1$. We note that in the case of $n = 2$ there exists a better algorithm achieving the optimal trade-off. See [67] for a detailed analysis.

3.3 Designing learning-augmented algorithms

It is now clear that we can use predictions to augment classical online algorithms so that we harness the power of accurate predictions and at the same time remain robust to inaccurate predictions. More examples of problems solved using predictions will be presented in the following chapters, however, here we want to point out a common pattern present in the design of almost all learning-augmented online algorithms. This pattern serves as a general method for designing learning-augmented online algorithms.

First, we have to decide what to predict. To answer this question we have to consider the offline optimal online algorithm. What is the parameter that enables the offline optimal algorithm to make its decisions? This is usually the parameter we want our predictor to predict. In the case of Ski rental this parameter was the number of skiing days, while in the case of Non-clairvoyant job scheduling this parameter was the processing times of the jobs.

Next we have to design the learning-augmented algorithm. Let $\mathcal{B}$ denote the algorithm following blindly the predictor. We bound the competitive ratio of $\mathcal{B}$ with respect to η. Finally, we combine $\mathcal{B}$ with the optimal online algorithm for the problem of interest. While both steps are usually problem specific, this surprisingly simple pattern seems to apply to almost all problems studied so far in the literature.

Chapter 4

Paging with predictions

In this chapter we are going to use predictions in order to augment classical online algorithms for the paging problem. This chapter is based on the work of Lykouris and Vassilvitskii [51], which initiated this line of research, Rohatgi [60], Wei [67] and the survey of Mitzenmacher and Vassilvitskii [57] .

The paging problem is a canonical example of online problems and has been extensively studied as we discussed in chapter 2 of this thesis. Specifically, we proved that any optimal deterministic algorithm for the paging problem achieves an $O(k)$ competitive ratio, while any optimal randomized algorithm is $O(\log k)$ competitive [31, 53, 1].

However, there is a well-known discrepancy between the theoretical performance of various algorithms (even with the same competitive ratio) and their empirical performance. For example, LRU outperforms almost all other algorithms (even randomized) in practice, due to the temporal and spatial locality of data present in almost all applications. Nonetheless, through the lens of worst-case analysis it was impossible to distinguish between these algorithms and explain their empirical performance since we had to be overly pessimistic about our algorithms and guard against any possible adversarial input. In practice, however, most instances of the paging problem follow a predictable pattern. Exploiting this pattern yields algorithms with better performance in most of the instances. Thus, the need for beyond-worst case analysis was born [61].

Such approaches include Parameterized algorithms [61](chapter 1), Resource augmentation [62], Bijective analysis [7], and many other beyond-competitive analysis approaches [46, 42].

The approach we examine in this thesis is that of learning-augmented algorithms. That is, along with the input of the problem we have access to an oracle predicting something useful about the input. How can we inform our decisions using the prediction?

Recall the paging problem. We have a slow memory of N pages and a fast memory (cache) of k pages. We have to answer a series of page requests. If the requested page is not in the fast memory (cache miss) we have to evict one of the k pages in cache to make space for the requested page. If the requested page is already in the cache (cache hit) we do not have to evict a page. The goal is to minimize the number of evictions.

In order to identify what the predictor should predict let us consider the optimal offline algorithm for the paging problem, which is known as Belady's rule [16]. Each time we have a cache miss the optimal strategy is to evict the page that will be requested furthest in the future. Thus, it will be logical that the predictor predicts the next time each page will be requested.

We will now proceed to formally define our model and present efficient learning augmented algorithms for the paging problem.

4.1 Paging with predictions

Assume that along with each requested page we have a prediction about the next time it will be requested again. Formally, let $next(x_i)$ be the next time page x_i will be requested. We

use $h(x_i)$ to denote the prediction for $next(x_i)$. We define the error as the ℓ_1-norm between the actual and the predicted values for each page. That is, $\eta = \sum |h(x_i) - next(x_i)|$.

Notice that the error grows with the input length. For example, if we duplicate a request sequence and the predictions, the error doubles but the competitive ratio of any reasonable algorithm would remain the same. One might be tempted to divide by the input length to fix this issue, however this also leads to a pathological case. One could increase the length of the sequence without changing the value of the optimum solution, or the impact of the predictions. For example, imagine after the last requested page we added many requests to the last page. All of these requests will be cache hits and the cost of any algorithm will remain the same. If the predictions are accurate the error will also remain constant but the normalized error will be smaller since the length of the sequence will be greater. Normalizing by the cost of the optimal algorithm addresses both of these problems. Thus, the critical parameter in bounding a learning-augmented algorithm for the paging problem is $\frac{\eta}{OPT}$, where OPT is the cost incurred by the optimal (offline) algorithm.

The fist approach one might try is to blindly trust the predictor and evict the page with the furthest in the future *predicted* next arrival time. We call this algorithm $\mathcal{B}$ since it tries to follow Belady's rule. If the predictions are accurate $\mathcal{B}$ is optimal, thus we have consistency. However, $\mathcal{B}$ is not robust. Formally, the competitive ratio of $\mathcal{B}$ is $\Omega(\frac{\eta}{OPT})$.

Consider the simple case of a 2-page cache and three elements a, b, c. Initially, we have in the cache pages a and c. The request sequence will be c, a, b, a, b, ..., a, b, c. The predictions for elements a and b will be correct, but the prediction for c will always be at time 0. Hence η is the length of the sequence. In this way, $\mathcal{B}$ will suffer a cache miss every time except for the last request. On the other hand, the optimal algorithm will only miss once in the last request. One might be tempted to fix the issue by evicting element (page c in our example) whose predicted times have passed. However, this algorithm has similarly bad performance.

4.1.1 Marking algorithms

For the first two learning-augmented algorithms we are going to present, we need to dive deeper into the analysis of Marking algorithms, which were presented in chapter 2 of this thesis. Marking algorithms form a family of algorithms working in a similar way. These algorithms run in phases. At the beginning of each phase all pages of the cache are considered "unmarked". When there is a cache miss an unmarked page is evicted and the requested element is "marked". When a cache hit occurs the element is also "marked". When all elements of the cache are marked and a cache miss occurs, a phase ends and we unmark all elements. The Marker algorithm evicts randomly an unmarked element.

We will partition the elements of each phase into two categories. We will say an element is *clean* in phase i if it appears in phase i, but not in phase $i-1$. If an element appears both in phase i and in phase $i-1$ it is called *stale*. The following two theorems relating the performance of the optimal and marker algorithm to the number of clean elements hold [31].

Theorem 4.1.1. *Let Q be the number of clean elements. Then the optimal algorithm suffers at least $\frac{Q}{2}$ misses.*

Theorem 4.1.2. *Let Q be the number of clean elements. Then the expected number of cache misses of the marker algorithm is $Q \cdot H_k$.*

4.2 Predictive Marker

We are now ready to examine the *Predictive Marker* algorithm proposed in [51]. We will describe the basic concepts of the algorithm and its analysis, but we will omit the rigorous proof of the results. The basic idea is the following.

We will use the predictor to decide which unmarked element to evict. If predictions prove inaccurate we are going to switch to the classical Marker algorithm evicting unmarked elements uniformly at random.

In order to analyse the *Predictive Marker* we have to understand the reason behind cache misses in terms of clean and stale elements. Suppose an element e is evicted during a phase. We will construct a blame graph as follows. Element e was evicted either because a clean element c arrived - we add a directed edge from e to c, or because a stale element s arrived, but s was previously evicted - we add a directed edge from e to s. This graph is a set of chains, which we call eviction chains. Every chain begins with a clean element and the rest of the elements are stale. The total length of the chains is equal to the total number of cache misses incurred by our algorithm. Thus, bounding the length of these chains will aid us bounding the overall cost of our algorithm.

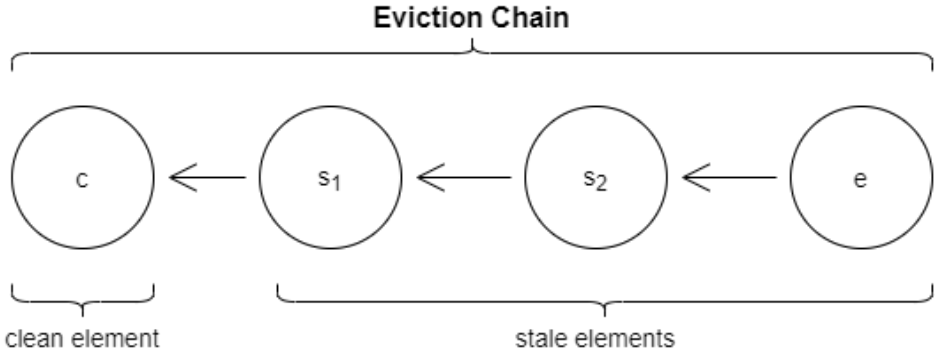

Figure 4.1: Eviction chain

When a stale element arrives we will evict a new element based on the prediction if the chain containing it has length less than H_k. Otherwise, we will evict an element uniformly at random following the classical Marker algorithm. Switching to the classical Marker algorithm can only augment a chain by at most H_k elements. One can prove that using this strategy the eviction chains have length $O(\min(\sqrt{\eta}, H_k))$. Using Theorem 4.1.1, we have constructed a learning-augmented algorithm achieving an $\min(2 + 4\sqrt{\frac{\eta}{OPT}}, 4H_k) = O(\min(\sqrt{\frac{\eta}{OPT}}, H_k))$ competitive ratio.

It turns out that a simpler approach suffices to obtain the same competitive ratio. We may trust the predictor only once at the beginning of each chain. That is, if the requested element is clean, evict the element with the furthest in the future predicted time of arrival. If the requested element is stale, evict an unmarked element uniformly at random. This modification of the *Predictive Marker* is called *LMarker* and is analysed in [60]. Furthermore, in [60] they design algorithm *LNonMarker* that achieves a competitive ratio of $O(1+\min(1, \frac{\eta}{kOPT}) \log k)$. In order to prove this competitive ratio for the *LNonMarker* a technique used and discussed in chapter 3 is utilized. That is, the combination of two online algorithms to get the best of both. This technique will also be used in the *BlindOracle* algorithm, in the following paragraph.

4.3 BlindOracle

We are now going to present the results of [67]. In this work, they prove that a surprisingly simple approach yields optimal results for deterministic algorithms. To be more precise, we can combine the algorithm blindly following the predictions, called *BlindOracle*, with the best online algorithm without predictions (for example LRU).

4.3.1 Combining online paging algorithms

The combination of multiple online algorithms was first considered in [31] as we discussed in chapter 2. In the special case of two online algorithms for the paging problem we have the following theorem.

Theorem 4.3.1. *Given any two algorithms $\mathcal{A}$ and $\mathcal{B}$ for the online caching problem, there exists an algorithm $\mathcal{C}$ such that:*

$$ALG_C(\sigma) \leq 2\min(ALG_A(\sigma), ALG_B(\sigma)) + O(1).$$

Algorithm $\mathcal{C}$ is actually very simple. We just use a "follow the leader" approach. That is, we simulate both algorithms (in parallel) and at each timestep we evict any page that is not in the cache of the better performing algorithm so far. We note that if algorithms $\mathcal{A}$ and $\mathcal{B}$ are deterministic, then $\mathcal{C}$ is also deterministic.

Furthermore, using a multiplicative weights approach one can design a randomized combinator [18].

Theorem 4.3.2. *Given any two algorithms $\mathcal{A}$ and $\mathcal{B}$ for the online caching problem, and any $0 < \epsilon < 1/4$, there exists an algorithm $\mathcal{C}$ such that:*

$$ALG_C(\sigma) \leq (1+\epsilon)\ \min(ALG_A(\sigma), ALG_B(\sigma)) + O(\epsilon^{-1}k).$$

4.3.2 Analysis of BlindOracle

We will now give a sketch of the analysis of the BlindOracle algorithm. First, we have to define the notion of inversions.

Definition 4. *We call a pair of elements (x_i, x_j) an* inversion *if $t_i < t_j$, but $h_i \geq h_j$, where t_i and h_i is the actual and the predicted time of next arrival for element x_i.*

We denote by $M = M(h,t)$ the total number of inversions until time t. In our analysis we will use the following lemma from [60]:

Lemma 4.3.3. *Let $\eta = \eta(h,t)$ be the total error and $M = M(h,t)$ the total number of inversions until time t.*

$$\eta \geq \frac{M}{2}.$$

Let A denote the optimal algorithm OPT and B denote the BlindOracle algorithm. We use A_t and B_t to denote the elements in the cache of the respective algorithms. Throughout our analysis we will maintain a matching $\mathcal{X}_t$ between A_t and B_t at all times. The matching $\mathcal{X}_t$ consists of pairs of elements $(a,b) \in A_t \times B_t$ such that the next arrival of b is no later than the next arrival of a. Intuitively, element b is "better" than element a since it will be requested later in time and thus algorithm A will incur a cache miss first (considering only those two elements).

We will now define a potential function $\Phi = \Phi(t) = \Phi(A_t, B_t, X_t)$ as the number of unmatched pages in B_t. Using the above definitions one can prove the following theorem.

Theorem 4.3.4. *There exists a valid matching X_t such that*

$$B + \Phi(t) \leq OPT + M.$$

The proof requires detailed case analysis and is omitted since it does not offer any further understanding. Using lemma 4.3.3 we have the following theorem:

Theorem 4.3.5. *The competitive ratio of $\mathcal{B}$ is at most $1 + \frac{\eta}{OPT}$.*

Using a different (and more complicated) potential function we can get a better dependence on the error. Specifically, one can prove the following theorem.

Theorem 4.3.6. *The competitive ratio of $\mathcal{B}$ is at most $2 + \frac{4}{k-1}\frac{\eta}{OPT}$.*

Thus, we have proven the following theorem.

Theorem 4.3.7. *For learning-augmented online caching,* BlindOracle *obtains a competitive ratio of*

$$\min\left(1 + 2\frac{\eta}{OPT}, 2 + \frac{4}{k-1}\frac{\eta}{OPT}\right).$$

Combining LRU with BlindOracle using theorem 4.3.1 we have the following theorem.

Theorem 4.3.8. *There exists a deterministic algorithm for learning-augmented online caching that achieves a competitive ratio of*

$$2\min\left(\min\left(1 + 2\frac{\eta}{OPT}, 2 + \frac{4}{k-1}\frac{\eta}{OPT}\right), k\right).$$

Finally, combining BlindOracle with an optimal randomized algorithm such as the Marker algorithm, we have the following theorem.

Theorem 4.3.9.

$$(1+\epsilon)\ \min\left(\min\left(1 + 2\frac{\eta}{OPT}, 2 + \frac{4}{k-1}\frac{\eta}{OPT}\right), 2H_k\right).$$

In [67] they prove that the competitive ratio of any deterministic algorithm for the learning-augmented online caching problem must be at least $1 + \Omega\left(\min\left(\frac{1}{k}\frac{\eta}{OPT}, k\right)\right)$. Thus, BlindOracle is optimal up to constant factors.

Once again, as in chapter 3, we were able to use the power of predictions to improve the classical online algorithms for the paging problem. It is important to note that the general technique that we presented in the conclusions of chapter 3 - i.e. the combination of the optimal online algorithm with the algorithm blindly following the predictor - stems from the work Wei [67] and the BlindOracle algorithm. We will see in the next chapters that this technique is ubiquitous in the field of learning-augmented online algorithms and can be applied to even more complicated problems.

In chapter 5 we are going to examine the paging problem as a special case of the more general metrical task systems problem [9]. In chapter 8 we are going to present a novel proof of the above bounds that circumvents the tedious case analysis of [67] and provides a deeper insight through the study of learning-augmented algorithms on matroid bases.

Furthermore, learning-augmented analysis has also been applied to the more general problem of weighted caching [40].

Finally, a worth reading comparative study of all known learning-augmented algorithms for the paging problem based on real-world datasets can be found in [27].

Chapter 5

Metrical Task Systems with predictions

In this chapter we will develop learning-augmented algorithms for the *Metrical task systems (MTS)* problem [20]. This chapter is based on the work of Antoniadis, Coester, Elias, Polak and Sinon [9]. As we discussed in chapter 2 of this thesis, the MTS problem generalizes many fundamental online problems, including caching, list-accessing and k-server, and has many practical applications - see [64, 52, 28, 39].

Recall the problem. We are given a metric space M of states. We start at an initial state x_0. At each timestep $t = 1, 2, ...$ we are given a task $\ell_t : M \to \mathbb{R} \cup \{0, +\infty\}$. We have to decide whether to stay at x_{t-1} and pay $\ell_t(x_{t-1})$ or move to another state x_t and pay $dist(x_{t-1}, x_t) + \ell_t(x_t)$, where $dist(x_{t-1}, x_t)$ is the cost of moving from state x_{t-1} to state x_t. The goal is to minimize the overall cost.

Once again the difficulty of the problem lies in the uncertainty about the future. We do not know neither the future tasks nor the underlying costs for staying at each state since these costs might change over time. What prediction would be helpful in this setting?

To answer this question we usually consider what the optimal offline algorithm does. However, in the case of MTS the optimal offline algorithm is not as simple as it was in the previous problems we tackled such as the ski rental problem or the paging problem. The optimal algorithm is not a greedy algorithm that makes decisions based on a simple parameter of the input that could be predicted. On the contrary, it has to consider the entire time horizon and have knowledge of the entire instance to make optimal decisions. Thus, the prediction of a simple parameter of the problem will not suffice. We have to consider a different approach.

Imagine we have a predictor predicting the next state we have to move to. Ideally, if the predictor is accurate it will predict all the states that the optimal algorithm selects and thus our algorithm will also be optimal. The error at time t will be the distance between the optimal state o_t and the predicted state p_t. In this way, we have embedded the intricacies of the optimal algorithm into the predictor.

Now we are ready to apply our general strategy. We will analyze the cost of the algorithm following the predictor (although this time with some caution) and then combine it with the best online algorithm to achieve the best of both.

Formally, the error η is defined as follows:

$$\eta = \sum_{t=1}^{T} \eta_t, \ \eta_t = dist(p_t, o_t),$$

where o_t are the states of some offline algorithm OFF.

Note that the offline algorithm OFF can be the optimal algorithm, but it can also be a near-optimal algorithm or a heuristic that performs well in practice.

Furthermore, we will normalize the error η with the cost of the offline algorithm OFF, as we discussed in the previous chapter.

Note that even if the error is low, the cost of the solution following the predicted states $p_1, p_2, ..., p_T$ may be much higher than the cost of OFF, since $\ell_t(p_t)$ can be much larger than

$\ell_t(o_t)$, even if $dist(p_t, o_t)$ is small. This is why we have to twist our algorithm and follow the predictor with caution.

5.1 Follow the Prediction

We will now formally define the *Follow the Prediction (FtP)* algorithm and analyze its performance. Let X be the set of all the states in our MTS. We define the *Follow the Prediction (FtP)* as follows: at time t, after receiving task ℓ_t and prediction p_t, FtP moves to state

$$x_t \leftarrow \arg\min_{x \in X} \{\ell_t(x) + 2dist(x, p_t)\}.$$

Intuitively, we trust the predictor unless it is better to move from the predicted state to another state, pay the service, and return back to the predicted state. Notice that in the case of uniform costs $\ell_t(x)$, we always follow the predictor.

We will prove the following lemma regarding the performance of FtP.

Lemma 5.1.1. *Algorithm FtP achieves a competitive ratio of* $1 + \frac{\eta}{OFF}$*, where* η *is the prediction error with respect to OFF.*

Proof. We define A_t to be the algorithm which agrees with FtP in its first t states $x_0, x_1, ..., x_t$ and then agrees with the states of OFF $o_{t+1}, ..., o_T$. Notice that $cost(A_0) = OFF$ and $cost(A_T) = FtP$, where we have abused the notation and OFF and FtP denote the respective costs of the algorithms. We will prove that $cost(A_t) \leq cost(A_{t-1}) + 4\eta_t$, for each t, where $\eta_t = dist(p_t, o_t)$. Summing up over all timesteps $t = 1, 2, ..., T$ will give us

$$cost(FtP) \leq cost(OFF) + 4\eta.$$

The algorithms A_t and A_{t-1} are in the same configuration at each time except t, when A_t is in x_t, while A_{t-1} is in o_t. By the triangle inequality we have that

$$cost(A_t) \leq cost(A_{t-1}) + 2dist(o_t, x_t) + \ell_t(x_t) - \ell_t(o_t) \Rightarrow$$

$$cost(A_t) \leq cost(A_{t-1}) + 2dist(o_t, p_t) - \ell_t(o_t) + 2dist(p_t, x_t) + \ell_t(x_t)$$

Using the definition of x_t we have that

$$2dist(p_t, x_t) + \ell_t(x_t) \leq 2dist(p_t, o_t) + \ell_t(o_t)$$

Thus, we have

$$cost(A_t) \leq cost(A_{t-1}) + 4dist(o_t, p_t)$$

□

Combining algorithm FtP with an online algorithm A with competitive ratio a, using the results techniques for the online combination of algorithms presented in chapter 2, we get the following theorem.

Theorem 5.1.2. *Let A be a deterministic a-competitive online algorithm for an MTS problem P. There is a learning-augmented deterministic algorithm for P achieving competitive ratio*

$$9 \cdot \min\{a, 1 + \frac{4\eta}{OFF}\}$$

against any offline algorithm OFF, where η *is the prediction error with respect to OFF.*

One can prove that this bound is tight up to constant factors (see theorem 9 in [9].

5.2 Caching

For some specific instances of MTS the dependence on $\frac{\eta}{OFF}$ can be improved. In particular in the case of caching, one can achieve logarithmic dependence on $\frac{\eta}{OFF}$. In [9] they develop an algorithm, which they call TRUST&DOUBT that achieves competitive ratio of $O(\min\{1 + \log(1 + \frac{\eta}{OFF}), \log k\})$. The interesting characteristic of TRUST&DOUBT, which is unique compared to previous approaches, is that it is able to gradually adapt the level of trust in the predictor throughout the instance. For a detailed analysis of TRUST&DOUBT we refer the reader to section 4 of [9].

Furthermore, in [9] they prove that the predictions used in the traditional learning-augmented framework for paging are not sufficient for more general problems such as the weighted caching problem. That is, predicting the next arrival time of each element does not yield better algorithms than the classical online algorithms without predictions. The learning-augmented weighted paging problem is considered in [40].

In chapter 8, we are going to explore a more general problem than caching which is also a special case of MTS. Namely, we will examine the problem of maintaining matroid bases with uniform costs and show that the traditional framework of caching can generalize to that case.

Chapter 6

The Primal-Dual method with predictions

In this chapter we are going to examine the primal-dual method, which is a powerful technique widely used in the design of online algorithms. Then, using predictions we will augment this method to the learning-augmented primal-dual method.

The primal dual method has been successfully used to tackle various problems such as the ski rental problem, the online set cover problem [4], the paging problem [13], the routing problem [24], network optimization problems [3] and many others. For a detailed analysis see [25].

In the learning-augmented setting we will present a novel approach different from what we have encountered so far. To be more precise, the predictor will not predict a single parameter of the instance nor the states of the optimal algorithm. Instead, it will predict the entire solution for the given instance. While this prediction might seem unrealistic it is abstract enough to capture numerous problems. Moreover, experimental results suggest its applicability to real-world problems. This chapter is based on the work of Bamas, Maggiori and Svensson [12].

6.1 The primal-dual method

First, we are going to demonstrate the primal-dual method in the standard case of online algorithms without predictions. This section is based on the results of [25]. For a gentle introduction the read is encouraged to see chaptes 1-3 of [25]. In this section we will state the basic results presented in chapter 4 of [25] concerning the online set cover problem.

In the set cover problem we have a universe of elements $\mathcal{U}$ and a collection of sets $\mathcal{S}$. Each set in $\mathcal{S}$ covers some elements of $\mathcal{U}$ and has an acquisition cost. Our goal is to cover all elements of $\mathcal{U}$ while minimizing the total cost.

In the online version of the problem, the elements of $\mathcal{U}$ are revealed in an online fashion. That is, at each timestep a new element is revealed along with the sets that cover it. The objective is again to cover all elements while minimizing the total cost.

The set cover problem is a paradigm covering problem and can be formulated as an LP as follows.

The primal linear program is the following.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} c_i x_i \\ \text{subject to} \quad & \sum_{i \in S(j)} x_i \geq 1, \forall j \\ & x_i \geq 0, \qquad \forall i \end{aligned}$$

The corresponding dual program is the following.

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{m} y_j \\ \text{subject to} \quad & \sum_{j|i\in S(j)} y_j \leq c_i, \forall i \\ & y_j \geq 0, \qquad \forall j \end{aligned}$$

In the online setting the constrains of the primal program are given one-by-one as new elements are revealed. Moreover, any valid online algorithm is only allowed to increase the variables x_i. In the online packing problem the values c_i are not known in advance. In the j-th round a new variable y_j is introduced to the algorithm, along with the set of packing constraints it appears in. Moreover, the algorithm may increase the value of y_j only in the round it is first given. This means that each packing constraint is gradually revealed to the algorithm.

Notice that we consider fractional solutions of the set cover problem. Having a fractional solution we can apply rounding techniques to get an integral one [2].

We will now present an online algorithm for the set cover problem.

Let $S(j)$ be the sets that cover element x_j.

Algorithm 7

Whenever a new primal constraint $\sum_{i\in S(j)} x_i \geq 1$ and the corresponding dual variable y_j appear:
while $\sum_{i\in S(j)} x_i < 1$ **do**
for each $i \in S(j)$: $x_i \leftarrow x_i(1 + \frac{1}{c_i}) + \frac{1}{|S(j)|c_i}$
$y_j \leftarrow y_j + 1$
end while

We will assume that each $c_i \geq 1$. Moreover, let $d = \max_j |S(j)| \leq m$ be the maximum number of sets that cover a single element. We prove the following theorem.

Theorem 6.1.1. *Algorithm 7 produces:*

- *A fractional covering solution which is $O(\log d)$ competitive.*
- *An integral packing solution which is 2-competitive and violates each packing constraint by at most a factor of $O(\log d)$.*

Proof. Let P and D be the values of the objective function of the primal and the dual solution the algorithm produces respectively. Initially, $P = D = 0$. Let ΔP and ΔD be the changes in the primal and dual cost, respectively, in a particular iteration of the algorithm. We prove the following claims:

1. The algorithm produces a primal (covering) feasible solution.
2. In each iteration: $\Delta P \leq 2\Delta D$.
3. Each packing constraint in the dual program is violated by at most $O(logd)$.

Proof of (1): Consider a primal constraint $\sum_{i\in S(j)} x_i \geq 1$. During the j-th iteration the algorithm increases the values of the variables x_i until the constraint is satisfied. Subsequent increases of the variables cannot make the solution infeasible.

Proof of (2): Whenever the algorithm updates the primal and dual solutions, the change in the dual profit is 1. The change in the primal cost is

$$\sum_{i \in S(j)} c_i \Delta x_i = \sum_{i \in S(j)} c_i \left(\frac{x_i}{c_i} + \frac{1}{|S(j)|} \right) = \sum_{i \in S(j)} \left(x_i + \frac{1}{|S(j)|} \right) \leq 2.$$

Proof of (3): Consider any dual constraint $\sum_{j|i \in S(j)} y_j \leq c_i$. Whenever we increase some y_j such that $i \in S(j)$ be one unit we also increase the variable x_i. Thus, variable x_i is bounded from below by the sum of a geometric sequence with $a_1 = \frac{1}{dc_i}$ and $r = (1 + \frac{1}{c_i})$. That is,

$$x_i \geq \frac{1}{d} \left(\left(1 + \frac{1}{c_i} \right)^{\sum_{j|i \in S(j)} y_j} - 1 \right). \tag{6.1}$$

Next, observe that the algorithm never updates any variable $x_i \geq 1$, since it cannot be in an unsatisfied constraint. We have that

$$x_i \leq x_i(1 + \frac{1}{c_i}) + \frac{1}{dc_i} < 1(1 + 1) + 1 = 3,$$

since $c_i \geq 1$ and $d \geq 1$.

Using inequality 1 and solving for $\sum_{j|i \in S(j)} y_j$ we have that

$$\sum_{j|i \in S(j)} y_j \leq c_i \log(3d + 1) = c_i O(\log d).$$

□

6.2 Learning-augmented primal-dual method

In this section we are going to extend algorithm 7 using predictions. This section is based on the results of Bamas, Maggiori and Svensson [12].

Formally, let $\mathcal{A}$ be a predicted solution, $\mathcal{I}$ the instance we are solving, a robustness parameter λ and $c_{ALG}(\mathcal{A}, \mathcal{I}, \lambda)$ the output of our algorithm. Intuitively, the robustness parameter captures our trust in the predicted solution. As $\lambda \to 0$ our trust in the predictor grows. We indicate by $S(\mathcal{A}, \mathcal{I})$ the cost of the output solution on input $\mathcal{I}$ if our algorithm blindly follows the prediction $\mathcal{A}$.

We will now define the concepts of consistency and robustness in this setting. Given a robustness parameter $0 < \lambda \leq 1$, we say that algorithm ALG is $C(\lambda)$-consistent and $R(\lambda)$-robust if the cost of the output solution satisfies:

$$c_{ALG}(\mathcal{A}, \mathcal{I}, \lambda) \leq \min\{C(\lambda) \cdot S(\mathcal{A}, \mathcal{I}), R(\lambda) \cdot OPT(\mathcal{I})\}.$$

If the prediction $\mathcal{A}$ is accurate and we trust the prediction ($\lambda \to 0$), we want our performance to be close to the optimal offline algorithm. On the other hand,if our confidence in the prediction is low ($\lambda \to 1$), then we want our algorithm to be close to the online optimal algorithm.

We are now ready to extend Algorithm 7 to take advantage of the prediction $\mathcal{A}$.

For simplicity we assume that the predicted solution is feasible - i.e. it covers all the elements. Formally, we have that $|S(j) \cap \mathcal{A}| \geq 1$, for every element e_j. The general idea of the extended algorithm is the following. If a set s_i is in the predicted solution $\mathcal{A}$, then its corresponding primal variable x_i is updated more aggressively. If the set s_i is not in the predicted solution, the update is more conservative. The primal-dual learning-augmented algorithm (PDLA) is the following.

Algorithm 8

Whenever a new element e arrives do the following:
while $\sum_{s_i \in S(e)} x_i < 1$ **do**
  for $s_i \in S(e)$ and $s_i \in \mathcal{A}$ **do** ▷ aggressive update
    $x_i \leftarrow x_i(1 + \frac{1}{c_i}) + \frac{\lambda}{c_i|S(j)|} + \frac{1-\lambda}{c_i|S(j) \cap \mathcal{A}|}$
  end for
  for $s_i \in S(e)$ and $s_i \notin \mathcal{A}$ **do** ▷ conservative update
    $x_i \leftarrow x_i(1 + \frac{1}{c_i}) + \frac{\lambda}{c_i|S(j)|}$
  end for
end while

For the performance of Algorithm 8 the following theorem holds.

Theorem 6.2.1. *Assuming $\mathcal{A}$ is a feasible solution, the cost of the fractional solution output by Algorithm 8 satisfies*

$$c_{PDLA}(\mathcal{A}, \mathcal{I}, \lambda) \leq \min\{O\left(\frac{1}{1-\lambda}\right) \cdot S(\mathcal{A}, \mathcal{I}), O\left(\log\left(\frac{\lambda}{d}\right)\right) \cdot OPT(\mathcal{I})\}.$$

The proof of this theorem is very similar to the proof for the classical online algorithm. To prove robustness we mimic the original proof for algorithm 7. To prove consistency we split the primal increase into ΔP_c which denotes the primal increase due to the aggressively updated sets (i.e. sets in the predicted solution $\mathcal{A}$) and ΔP_u for the sets not in the predicted solution. Thus, $\Delta P = \Delta P_u + \Delta P_c$. Then we prove that $\Delta P \leq O\left(\frac{1}{1-\lambda}\right) \Delta P_c$. Since ΔP_c is due to the predicted sets, we can charge this increase to $S(\mathcal{A}, \mathcal{I})$. We refer the reader to [12] for the full proof.

6.3 Applications

The learning-augmented primal-dual method can be used to solve the ski rental problem, the Bahncard problem [33] and the dynamic TCP acknowledgement problem [41]. See [12] for a detailed analysis.

We are now going to present an application of the learning-augmented primal-dual method to the Parking Permit Problem [54]. The parking permit problem is in the broad category of leasing problems, see [59] and [8]. Relevant is the buy-at-bulk problem [11]. Moreover, the parking permit problem can be seen as a generalization of the ski rental problem where we can rent the skis for different periods of time, instead of just having to either rent for one day or buy them.

6.3.1 The Parking permit problem

Imagine that on certain days we have to drive to work. Whether we will drive to work is not known in advance since it is influenced by unknown factors such as the weather or our employer's orders. If we drive to work, we have to buy a parking permit in order to park our car. There are many available permits for different periods of time, for example daily, weekly, monthly, yearly. Of course the permits for the longer periods of time are cheaper per day. Our goal is to minimize the total cost for buying parking permits.

Formally, we are given K different types of permits which we can purchase. Permit k has cost C_k and duration D_k. Its day we are announced whether we have to drive to work or not. The goal is to minimize the total cost.

We can consider an equivalent model of the problem, which we call the interval model, where each permit is only available during a specific time period. Moreover, we partition the

entire time horizon T into smaller intervals of time D_k for each type of permit k. Thus, each time is covered by exactly K different permits.

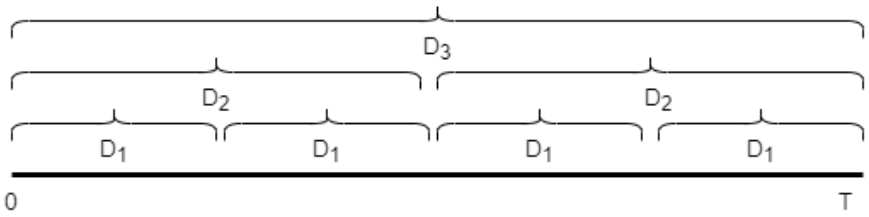

Figure 6.1: An instance of the parking permit problem with 3 different types of permits in the interval model.

Using this model it is obvious that every instance of the parking permit problem can be seen as an instance of the online set cover problem. Driving days are the "elements" that we have to cover and the permits are the sets. Each driving day should be covered by a set - i.e. permit. Thus, the results of the previous section also hold for the Parking permit problem. Note that any fractional algorithm for the parking permit problem gives rise to a randomized algorithm with expected cost within $\Theta(1)$ of the fractional cost (see section 4 of [54]).

6.3.2 Extensions to leasing problems on graphs

The result can be further extended to the following variant of the online steiner forest problem as discussed in section 5 of [54]. In this problem we are given weighted graph $G = (V, E)$. Pairs of communicating nodes announce themselves over time and we have to connect them via a path. Our algorithm is allowed to rent edges for different periods. Similarly to the parking permit problem we are given K such types of leases where lease i has cost C_i and duration D_i. We pay $C_i w(e)$ to rent edge e for time D_i.

If G is a tree then each pair of nodes is connected via a unique path and thus we have to the rent the edges of this path. Using the same approach as the algorithm for the parking permit problem we can again formulate the problem as a covering problem. In this case the elements will pairs of days and edges. That is, if edge e is in a path connecting two communicating nodes that were announced at day t, we have to "cover" edge e for all the days after day t. Thus, the learning-augmented algorithm still applies for this variant of the steiner tree problem.

Finally, these results can be further extended in the case where G is any graph using the results of [30] incurring a factor of $O(log|V|)$ in the competitive ratio.

An interesting research direction would be to apply the learning-augmented primal-dual method to other covering problems as well as packing problems such as revenue maximization in ad-auctions [23].

Chapter 7

Job-Scheduling revisited

In chapter 3, we introduced the non-clairvoyant job-scheduling problem and provided a framework to incorporate predictions. Let us recall the problem. We have a set of jobs that need to be scheduled on a single machine with the goal of minimizing the total completion time of all jobs. The jobs sizes are not known in advance. It is a well-known fact, that the Round-Robin (RR) algorithm that runs all the jobs in equal portion in a cyclical fashion is the optimal online algorithm. The optimal offline algorithm is the Shortest-Job-First (SJF) algorithm that schedules the jobs in non-decreasing order of their sizes.

In chapter 3, we saw that combining the Shortest-Predicted-Job-First (SPJF) algorithm with the Round-Robin (RR) algorithm yields an algorithm that satisfies both of the two desiderata we have for any learning-augmented algorithm. That is, if the predictions are perfect, our algorithm performs close to the optimal offline algorithm (SJF) and if the predictions are inaccurate our algorithm performs close to the optimal online algorithm (RR).

In this chapter, we will define a new property that we want our error function to satisfy. Then, we will define a new error function for the job-scheduling problem that satisfies that property. This chapter is based on the recent work of Im, Kumar, Purohit, Qaem [37].

In the previous setting, the oracle predicted the sizes of the jobs and the error was defined as the sum of the absolute difference of the predicted and true sizes of the jobs. As we will see, this formulation fails to distinguish between good and bad predictions since it might be the case that a larger error yields a better solution or that a smaller error yields a worse solution. Moreover, small perturbations in the predictions might lead to large changes to the optimal solution. We will address these problems by introducing a new Lipschitz-like property that we want our error function to satisfy.

7.1 Prediction error

Formally, let J be the set of all jobs and $\{p_j\}_{j\in J}$ and $\{p'_j\}_{j\in J}$ be the actual and the predicted sizes of the jobs. We want our error $ERR(\{p_j\}_{j\in J}, \{p'_j\}_{j\in J})$ to satisfy the following properties.

Property 3 (Monotonicity)**.** *For any $I \subseteq J$,*

$$ERR(\{p_j\}_{j\in J}, \{p'_j\}_{j\in J\setminus I} \cup \{p_j\}_{j\in I}) \leq ERR(\{p_j\}_{j\in J}, \{p'_j\}_{j\in J})$$

That is, if more job sized predictions are correct, then the error must decrease. This property was also true in the framework of chapter 3.

we will now define a Lipschitz-like property stating that the optimal solution of the predicted instance should be close to the optimal solution of the original solution if and only if the prediction is "good". To be more precise, if $|OPT(\{p_j\}_{j\in J}) - OPT(\{p'_j\}_{j\in J})|$ is large, then we must have a large error.

Property 4 (Lipschitzness)**.**

$$|OPT(\{p_j\}_{j\in J}) - OPT(\{p'_j\}_{j\in J})| \leq ERR(\{p_j\}_{j\in J}, \{p'_j\}_{j\in J}).$$

For simplicity, we write $p = \{p_j\}_{j \in J}$. The prediction error defined in chapter 3, $\ell_1(p, p') = \sum_j |p_j - p'_j|$ does not satisfy the Lipschitz property. Let the true job size be $p_1 = 1 + \epsilon$ and $p_j = 1, j \in J \setminus \{1\}$. Now consider the following two predictions. Let $p'_1 = 1 + 3\epsilon$ and $p'_j = 1, j \in J \setminus \{1\}$. Let $q'_1 = 1 - \epsilon$ and $q'_j = 1, j \in J \setminus \{1\}$. For the ℓ_1 error it holds that $\ell_1(p, p') = 2\epsilon = \ell_1(p, q')$. However, we have that $OPT(p') - OPT(\{p\}) = 2\epsilon$, while $OPT(p) - OPT(q') = (n + 1)\epsilon$. Intuitively, the predictions q' are much worse than p', but subjected to the ℓ_1 error it is impossible to distinguish them.

One might be tempted to define the error as $ERR(p, p') = |OPT(p) - OPT(p')|$, but this error does not satisfy the monotonicity property. Imaging predicting a permutation of the true job sizes i.e. the job sizes are accurate but correspond to different jobs. In this case, $|OPT(p) - OPT(p')| = 0$, but improving any of the predictions will lead to a different optimum value and thus greater error.

We will now define a measure of error that satisfies all the required properties.

Definition 5 (Prediction Error)**.** *For any instance of the non-clairvoyant scheduling problem with predictions the prediction error is defined as*

$$v(J, p, p') = OPT(\{p'_j\}_{j \in J_o} \cup \{p_j\}_{j \in J_u}) - OPT(\{p_j\}_{j \in J_o} \cup \{p'_j\}_{j \in J_u}),$$

where $J_o = \{j \in J | p'_j > p_j\}$ and $J_u = \{j \in J | p'_j < p_j\}$ are the sets of jobs whose size is overestimated or underestimated respectively.

The above definition arises from the following two inequalities that we want our error to satisfy.

$$v \geq OPT(\{p'_j\}_{j \in J_o} \cup \{p_j\}_{j \in J_u}) - OPT(\{p_j\}_{j \in J_o} \cup \{p_j\}_{j \in J_u}),$$

if the underestimated job sizes were predicted correctly.

$$v \geq OPT(\{p_j\}_{j \in J_o} \cup \{p_j\}_{j \in J_u}) - OPT(\{p_j\}_{j \in J_o} \cup \{p'_j\}_{j \in J_u}),$$

if the overestimated job sizes were predicted correctly.

If we add the RHS of these inequalities, we get our error measure.

We will now briefly discuss the error measure we used in chapter 6. The error can be defined as the cost of Shortest Predicted Job First (SPJF) algorithm minus the cost of the optimal algorithm. It is clear that this error is neither monotone nor Lipschitz, since SPJF produces an optimal solution if and only if the order of the predicted jobs is accurate.

Finally, we note that the error measure has to consider job identities since the cost of the optimal algorithm will be the same on every permutation of the true job sizes, but the error will change.

7.2 Job-scheduling with predictions

In this section we will present the learning-augmented algorithm proposed in [37]. The algorithm works in rounds. Let J_k be the set of remaining jobs at the beginning of round k, $n_k := |J_k|$ and $q_{k,j}$ the amount of processing done on job j in round k. We also define $p_{k,j}$ and $p'_{k,j}$ as the true and the predicted remaining size of job j at the beginning of round k.

The algorithm makes use of the following procedure to estimate the median m_k of the true remaining size of jobs in J_k.

Algorithm 9 Median-estimator(J_k,δ, n)

Let S be a uniform random sample, with replacement, of size $\frac{\ln 2\delta}{\delta^2}$.
Run Round-Robin on S until half of the jobs in S complete; let j_k be the job that completed last.
Return $\tilde{m}_k = p_{k,j_k}$.

Furthermore, the algorithm makes use of the following procedure to estimate if the predictions are accurate for the remaining jobs in round k.

Algorithm 10 Error-estimator(J_k,ϵ, n, $\tilde{m}_k$)

Let P be a uniform random sample, with replacement, of size $\frac{1}{c}\log n$ from a family $Q := \{(j,j)|j \in J_k\} \cup \{(i,j)|i < j \in J_k\}$ of unordered pairs.
For every sampled job j, calculate $d_{k,j}$ by running j up to $(1+\epsilon)\tilde{m}_k$ units.
Return $\tilde{\eta}_k := |Q|\frac{1}{|P|}\sum_{(i,j)\in P}\min\{d_{k,i}, d_{k,j}\}$.

The main idea of the algorithm is the following. If we have enough jobs remaining, we estimate the median and the error. If the error is big, the we run the Round-Robin algorithm. We process each job at most $2\tilde{m}_k$ units. If the error is small, we run the SPJF algorithm. We process each job at most $3\epsilon\tilde{m}_k$ units. Finally, when we do not have enough jobs for our estimators, we run the Round-Robin algorithm to complete all remaining jobs.

We are now ready to present the algorithm for the scheduling problem with predictions.

Algorithm 11 Scheduling with predictions

$k \leftarrow 1$ and $\delta \leftarrow \frac{1}{50}$.
while $n_k \geq \frac{1}{c^3}\log n$ **do**
  $\tilde{m}_k \leftarrow Median - estimator(J_k, \delta, n)$
  $\tilde{\eta}_k \leftarrow Error - estimator(J_k, \epsilon, n, \tilde{m}_k)$
  if $\tilde{\eta}_k \geq \epsilon\delta^2\tilde{m}_k n_k^2/16$ **then** ▷ RR (big error) round
    Process each job in J_k up to $2\tilde{m}_k$ units with Round-Robin.
  else ▷ Non-RR (small error) round
    Process jobs j with $p'_{k,j} \leq (1+\epsilon)\tilde{m}_k$ up to $p'_{k,j} + 3\epsilon\tilde{m}_k$ units in increasing order of $p'_{k,j}$.
  end if
  $k \leftarrow k+1$
end while
Run Round-Robin to complete the remaining jobs ▷ Round K+1

The analysis of the algorithm is fairly complicated and is omitted. We refer the reader to [37] for a detailed analysis.

We quote the following result regarding the performance of the above algorithm.

Theorem 7.2.1. *Algorithm's 11 objective is at most* $\min\{O(1)OPT, (1+\epsilon)OPT+2OPT(J_{K+1})+O(1/\epsilon^2)v\}$ *with high probability, where* $|J_{K+1}| \leq \frac{1}{\epsilon^3}\log n$.

That is, Algorithm 11 is $O(1)$-robust and $(1+\epsilon)$-consistent, for any $\epsilon > 0$ with high probability, if no subset of $O(\frac{1}{\epsilon^3}\log n)$ dominates the objective.

It would be interesting to apply the above framework to other problems and see how previous results might change. We note, however, that although the Lipschitz property is desirable, it complicates the analysis and the design of learning-augmented algorithms.

Chapter 8

Changing Bases with predictions

In most combinatorial optimization problems one is concerned with solving an instance frozen in time and usually the goal is to minimize the total cost incurred. However, in real-world problems one has to solve an instance of the same problem repeatedly. Moreover, the underlying costs might change over time. One approach is to resolve the new instance from scratch, however, in most cases there is a transition cost between the old and the new solution. Thus, it might be better to start with the old solution and carefully transform it to a solution for the new instance.

Imagine having to solve the minimum spanning tree problem in a graph where at each timestep some edges died and some new edges were born. Finding the minimum spanning tree at each timestep separately may lead to more costly solutions. It might be better to retain the spanning tree of the previous timestep and add the necessary edges to complete the new spanning tree. It is thus a game of balance between acquiring new edges and retaining the edges of the old spanning tree. This is the problem we are to going to examine in this chapter.

More formally, we will examine the *Multistage Matroid Maintenance problem (MMM)*, where the underlying structure is a base of a matroid. We will formally define what a base of a matroid is in the next section. For now, it suffices to think that a matroid base is a generalization of the notion of the spanning tree of a graph. We encourage the reader to think in terms of spanning trees of graphs (or graphical matroids) in order to intuitively grasp the ideas we are going to present.

In this chapter, we will briefly examine the main results of [35] and then design learning-augmented algorithms for the MMM problem. We will also see how we can obtain previous results in the learning-augmented paging problem, through a more intuitive and simple approach.

Related work is that of [22], where they consider unified approached between Online learning and Competitive analysis to solve the problem of Metrical task systems. Furthermore, the work of [65], where they consider re-optimization problems - i.e. how one can start with an initial solution and adjust it to a new instance balancing the transition costs and the cost of the instance. Moreover, there related work is that of [36] where they examine algorithms for maintaining spanning trees in dynamic graphs. Along these lines is the problem of dynamic Steiner tree maintenance [38, 34].

8.1 Preliminaries

8.1.1 Matroids

Matroids are abstract structures generalizing the notion of linear independence in vector spaces. Formally, they are defined as follows: a finite matroid $\mathcal{M}$ is a pair $(E, \mathcal{I})$, where E is a finite set (called the ground set) and $\mathcal{I}$ is a family of subsets of E (called the independent sets) with the following properties:

- The empty set is independent, i.e., $\emptyset \in \mathcal{I}$.

- Every subset of an independent set is independent, i.e., for each $\mathcal{A}' \subseteq \mathcal{A} \subseteq E$, if $\mathcal{A} \in \mathcal{I}$, then $\mathcal{A}' \in \mathcal{I}$. (*hereditary property*)
- If $\mathcal{A}$ and $\mathcal{B}$ are two independent sets (i.e., each set is independent) and $\mathcal{A}$ has more elements than $\mathcal{B}$, then there exists $x \in \mathcal{A} \setminus \mathcal{B}$ such that $\mathcal{B} \cup \{x\}$ is in $\mathcal{I}$. (*augmentation property*)

A maximal independent set - that is, an independent set that becomes dependent on adding any element of E - is called a *basis* for the matroid. A set $S \subseteq E$ such that a base is subset of S is called a *spanning set*. A circuit in a matroid M is a minimal dependent subset of E - that is, a dependent set whose proper subsets are all independent. In graphs (graphical matroids), bases are spanning forests (or spanning trees, if the graph is connected) and circuits are simple circles.

Using the axioms of the matroids one can prove that any two bases of a matroid have the same number of elements. This number is called the rank of the matroid $\mathcal{M}$. We define the rank of an arbitrary set $\mathcal{S}$ to be the cardinality of the maximum cardinality independent subset of $\mathcal{S}$. In a graphical matroid $G(V, E)$, the rank is $r = |V| - 1$.

For the following we assume basic familiarity with matroids. For a detailed examination of matroids we refer the reader to [63].

8.2 Multistage Matroid Maintenance (MMM)

In this paragraph we formally define the problem of *Multistage Matroid Maintenance* (MMM) and briefly examine the results of [35].

8.2.1 The problem

Given reals $c(e)$ for elements $e \in E$ and a set $S \subseteq E$, we use $c(S)$ to denote $\sum_{e \in S} c(e)$. We use [T] to denote $\{1, 2, ..., T\}$.

An instance of the *Multistage Matroid Maintenance* (MMM) problem consists of a matroid $\mathcal{M} = (E, \mathcal{I})$, an *acquisition* cost $\alpha(e) \geq 0$ for each $e \in E$, and for every timestep $t \in [T]$ and element $e \in E$, a *holding* cost $c_t(e)$. The goal is to find bases $\{B_t \in \mathcal{I}\}_{t \in [T]}$ to minimize

$$\sum_t (c_t(B_t) + \alpha(B_t \setminus B_{t-1})),$$

where we define $B_0 = \emptyset$

An equivalent problem, as we will see in 8.2.1, is the *Multistage Spanning set Maintenance* (MSM) problem, where instead of a base we want to maintain a spanning set $S_t \subseteq E$ at each timestep. The cost of the solution $\{S_t\}_{t \in [T]}$ (with $S_0 = \emptyset$) is

$$\sum_t (c_t(S_t) + \alpha(S_t \setminus S_{t-1})).$$

We state a key lemma that will be useful both in the development of online algorithms for the MMM problem and in the learning-augmented analysis.

Lemma 8.2.1 (Maintaining Bases vs Maintaining Spanning Sets)**.** *For matroids, the optimal solutions to MMM and MSM have the same costs.*

Proof. Any solution to MMM is also a solution to MSM, since a base is also a spanning set. We will now show that a solution to MSM can be transformed to a solution to MMM with the same cost. Let $\{S_t\}$ be a solution to MSM. We set B_1 to any base in S_1. In order to construct B_t, we extend the (independent) set $B_{t-1} \cap S_t$ to any base B_t of S_t. This is possible

due to the matroid properties. Notice that this process can be performed online and thus the equivalence of MMM and MSM also holds in the online case.

Of course, the cost of $\{B_t\}$ is no more than that of $\{S_t\}$, since $B_t \subseteq S_t$. Moreover, let $D := B_t \setminus B_{t-1}$ be the elements we added at time t. Consider any element $e \in D$ and let $t^* \leq t$ be the time it was most recently added to the spanning set of MSM. The MSM solution paid for including e at time t^* and we charge our acquisition of e into B_t to this pair (e, t^*). Observe that we will not charge this pair again, since the procedure to create $\{B_t\}$ ensures we do not drop e from the base until it is dropped from S_t itself. □

8.2.2 The interval model

We now provide an equivalent definition of our problem: suppose each element e of the matroid has only an acquisition cost $\alpha(e)$ and is available only during an interval $I_e = [l_e, r_e]$. There are no holding costs. When an element's interval starts, we say that the element is born. Accordingly, when an element's interval ends, we say that the element dies. This model, which we call the interval model, is equivalent to the original model.

- *Offline (exact) reduction.* Given an instance of the MSM problem, create (parallel) elements e_{lr} for each $e \in E$ and $1 \leq l \leq r \leq T$, with acquisition cost $\alpha(e_{lr}) = \alpha(e) + \sum_{t=l}^{r} c_t(e)$ and interval $I_{e_{lr}} = [l, r]$.
- *Online (approximate) reduction.* For each element $e \in E$ define $t_0 = 0$. Create parallel copies $\{e_i\}$ such that the copy e_i has interval $I_{e_i} = [t_{i-1}+1, t_i]$, where t_i is set to $t_{i-1}+1$ if $c_{t_{i-1}+1}(e) \geq \alpha(e)$, else it is set to the largest time such that $\sum_{t=t_{i-1}+1}^{t_i} c_t(e) \leq \alpha(e)$. The acquisition cost of e_i is $\alpha(e_i) = \alpha(e) + c_{t_{i-1}+1}(e)$.

Notice that in the online reduction the intervals for an original element e partition the time horizon $[T]$ and thus at each timestep we have only $|E| = n$ elements alive.

8.2.3 Offline MMM

In order to develop efficient algorithms for the MMM problem which is a packing-covering problem it easier to use lemma 8.2.1 and instead consider the MSM problem which is a covering problem. One could use algorithms for submodular set cover [68] to get an $O(\log T)$ approximation. Furthermore, the greedy algorithm also yields an $O(\log T)$ approximation (see appendix B of [35]). In [35], they give an LP-rounding algorithm which is an $O(\log rT)$ approximation in the general case and an $O(\log r)$ approximation in the case of uniform costs. This algorithm will be extended to be used in the online instance of the problem. Finally, we state the following theorem regarding the hardness of approximating the MMM and MSM problems.

Theorem 8.2.2. *The MSM and MMM problems are NP-hard to approximate better than* $\Omega(\min\{\log r, \log T\})$ *even for graphical matroids.*

8.2.4 Online MMM

In the online setting the acquisition costs $a(e)$ are known up-front, but the holding costs $c_t(e)$ for time t are not known before time t. Recall that the equivalence between MMM and MSM still holds in the online setting and thus we can work with the MSM problem. In the interval model, at each time the elements whose time intervals have ended are announced. Moreover, the new elements along with their acquisition costs are announced. The goal is, again, to minimize the total cost.

In [35] they give an $O(\log |E| \log rT)$-competitive algorithm, by extending the LP-rounding algorithm used for the offline case. In the case of uniform costs the above ratio becomes

$O(\log |E| \log r)$. In chapter 8.5, we will use this algorithm to make our learning-augmented algorithm robust.

8.2.5 MMM with uniform costs

For the following, we assume that all elements of the matroid have unit costs and that at most one edge dies at each timestep. We will examine the problem without these constraints in section 8.8. Furthermore, at each timestep exactly $n = |E|$ elements are alive (see the online approximate reduction in the previous paragraph).

We use lemma 8.2.1 to prove the following lemma regarding the cost of the optimal algorithm when run on two bases differing in k elements.

Lemma 8.2.3. *Suppose we have two bases with the same elements expect for k elements, $0 \leq k \leq r$, where r is the rank of the matroid. We run the optimal algorithm on both instances (in parallel). We denote by $\mathcal{OPT}_1$ and $\mathcal{OPT}_2$ the execution of the optimal algorithm on the two instances respectively. Let OPT_1 and OPT_2 be the respective costs of the two solutions. Then,*

$$OPT_1 \leq OPT_2 + k.$$

Proof. Using lemma 8.2.1, $\mathcal{OPT}_1$ can form a spanning set buying the k different elements and be at least as good as $\mathcal{OPT}_2$. □

Now we extend lemma 8.2.3 for the case where instead of OPT we run an algorithm $\mathcal{A}$ which is c-competitive.

Lemma 8.2.4. *Suppose we have two bases with the same elements except for k elements. We run algorithm $\mathcal{A}$ on both instances (in parallel). We use A_1 to refer to the cost of executing $\mathcal{A}$ on the first instance and A_2 on the second instance. Then,*

$$A_1 \leq c\,(A_2 + k),$$

where c is the competitive ratio of $\mathcal{A}$.

Proof. We use OPT_1 and OPT_2 to refer to the cost of executing $\mathcal{OPT}$ on the first and the second instance respectively. We have that $A_1 \leq c\,OPT_1$ and $OPT_2 \leq A_2$. Moreover, from lemma 1.2 we have that $OPT_1 \leq OPT_2 + k$ and thus

$$A_1 \leq c\,(OPT_2 + k) \leq c\,(A_2 + k).$$

□

Despite their simplicity, the above lemmas will prove extremely useful in the learning-augmented setting.

8.2.6 The greedy algorithm

We will now define a natural greedy algorithm $\mathcal{G}$ for the MMM problem with uniform costs where at each time at most one element dies. The greedy algorithm is semi-online. That is, it makes decisions in an online fashion, but it has knowledge of the death times of all available elements. However, at each timestep t, $\mathcal{G}$ only considers elements available at this timestep and not elements that will be available in the future.

Algorithm $\mathcal{G}$ works as follows. Each time an element of our base dies, we buy the element dying furthest in the future such that our set remains independent. Note that the greedy algorithm is not optimal even in the case of graphical matroids.

It is easy to extend algorithm $\mathcal{G}$ in the case where multiple elements may die at each timestep. Namely, G will be the analogous of the famous Kruskal's algorithm for spanning

trees in graphs [48]. That is, we buy the edge dying furthest in the future such that our set remains independent. We continue this process until we have formed a base.

Algorithm $\mathcal{G}$ is known to be an $O(\log T)$ approximation of the optimal algorithm, even in the case of non-uniform acquisition costs. In the case of uniform costs, it is conjectured to be a constant approximation of the optimal algorithm (see appendix B in [35]). We do not know of an instance with uniform costs where $\mathcal{G}$ is not a constant approximation of the optimal algorithm. Furthermore, for different types of matroids a different bound may exist. For example in uniform matroids, $\mathcal{G}$ is optimal.

We will examine the cases where algorithm $\mathcal{G}$ is optimal.

8.3 Combining online algorithms

In this section we will restate the results of [9], which are in turn based on [32] and [18], that will enable us to combine multiple online algorithms and get the best of them. These are the same results we used in the previous chapter of this thesis.

Note that the MMM problem is a special case of the MTS problem. Let r be the rank of the underlying matroid in MMM. Imagine having a state s_i for each set of r elements. If the s_i corresponds to a valid base at time t, we have that $\ell_t(s_i) = 0$. Otherwise, $\ell_t(s_i) = +\infty$. The distance of two states can be defined as the cost of the distinct elements between the respective bases.

Theorem 8.3.1 (Deterministic combination)**.** *Given m online algorithms A_0, ..., A_{m-1} for an MTS, there exists a deterministic algorithm achieving cost at most $\frac{2\gamma^m}{\gamma-1} \cdot \min_i\{cost_{A_i}(I)\}$, for any input sequence I.*

Theorem 8.3.2 (Randomized combination)**.** *Given m online algorithms A_0, ..., A_{m-1} for an MTS with diameter D and $\epsilon < 1/2$, there is a randomized algorithm, such that for any instance I, its expected cost is at most*

$$(1+\epsilon)\ min_i\{cost(A_i(I))\} + O(D/\epsilon)\ln(m).$$

In our setting the diameter D of the metric space is the maximum cost for changing from one state to another. Since we will only consider the case of uniform unit costs $D = r$.

8.4 The learning-augmented setting

Suppose we have an oracle predicting the death time of each element. We define the *error* of the prediction $\eta = \sum_{e\in E} |t_e - h_e|$, where t_e and h_e are the actual and predicted death times of element e. We call a pair of elements (e_i, e_j) an *inversion* if $t_i < t_j$ but $h_i \geq h_j$. We denote by $M = M(h, t)$ the total number of inversions until time t. In our analysis we will use the following lemma from [60]:

Lemma 8.4.1. *Let $\eta = \eta(h, t)$ be the total error and $M = M(h, t)$ the total number of inversions until time t.*

$$\eta \geq \frac{M}{2}.$$

Let $\mathcal{B} = BlindOracle$ be the algorithm blindly following the predictor - that is, it buys (when needed) the element *predicted* to die furthest in the future.

8.5 Analysis of $\mathcal{B}$

Notation: We use $\mathcal{OPT}$ to denote the (offline) optimal algorithm. We use $\mathcal{G}$ to denote the (offline) greedy algorithm buying (when needed) the edge dying furthest in the future. Finally, we use $\mathcal{B}$ to denote the online algorithm buying (when needed) the edge *predicted* to die furthest in the future.

We will examine the case where $\mathcal{G}$ is optimal.

Theorem 8.5.1. *It holds that*

$$B \leq G + 2\eta,$$

where G is the cost of the greedy algorithm $\mathcal{G}$ and η the total error of the predictor.

Proof. We define A_t to be the (meta)algorithm running B for the first t timesteps and then running $\mathcal{G}$. Note that A_0 is the greedy algorithm $\mathcal{G}$ and A_T is algorithm B. We will prove that

$$A_t \leq A_{t-1} + M_t,$$

where we have abused the notation and used A_t to denote the cost of algorithm A_t. We note that A_t and A_{t-1} run the same policy expect for step t, when A_t runs B and A_{t-1} runs $\mathcal{G}$. The configurations (bases) and the costs of the two algorithms are the same until timestep t-1.

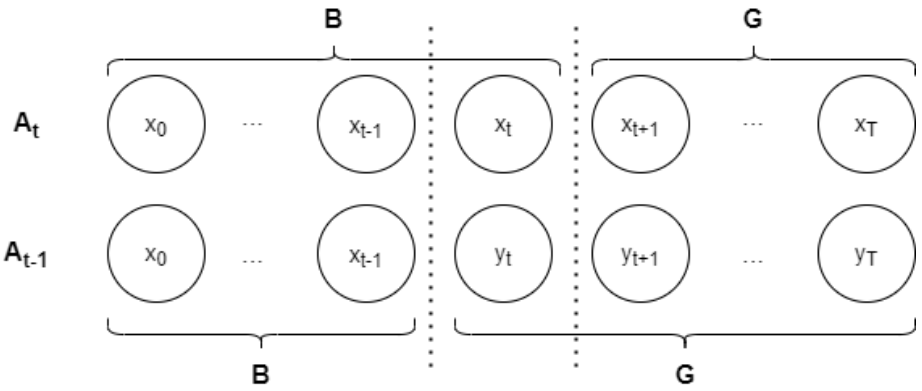

Figure 8.1: States of algorithms A_t and A_{t-1} and the respective algorithms run at each timestep

If no edge dies at timestep t, then the configurations will remain identical and thus $A_t = A_{t-1}$. If an edge dies at timestep t, then we have the following cases:

- $\mathcal{B}$ buys the edge dying furthest in the future: $A_t = A_{t-1}$, since both algorithms buy the same edge.
- $\mathcal{B}$ buys element e_1 while $\mathcal{G}$ buys element e_2: There exists an inversion between elements e_1 and e_2 and thus $M \geq 1$. After timestep t both A_t and A_{t-1} run algorithm $\mathcal{G}$. Using lemma 1.2 we have that $A_t \leq A_{t-1} + 1$. Charging the difference to the inversion we have that $A_t \leq A_{t-1} + M$.

Note that we do not double charge since if we buy an edge it cannot reappear in a future inversion. Summing over all timesteps t=0,1,...,T and using lemma 8.4.1 we have that

$$B \leq G + 2\eta.$$

□

For uniform matroids a better dependence on η is possible as we will prove in the following section. For general matroids the question remains open.

By combining algorithm $\mathcal{B}$ with an online algorithm A, using the results we presented in section 3, we have the following theorem.

Theorem 8.5.2. *Let A be an a-competitive online algorithm for the MMM problem with uniform costs. There is a learning-augmented online algorithm achieving competitive ratio*

$$O\left(\min\left(a, 1+\frac{2\eta}{OPT}\right)\right),$$

where c is the competitive ratio of $\mathcal{G}$ and OPT the optimal offline algorithm.

8.6 Uniform matroids

In the case of uniform matroids with uniform costs the greedy algorithm $\mathcal{G}$ is optimal. Thus, $c = 1$ and theorem 8.5.1 yields:

$$B \leq OPT + 2\eta.$$

Furthermore, we can obtain an asymptotically better (in k) bound. A more careful analysis shows that an upper bound with a $1/k$ coefficient on the ratio $\frac{\eta}{OPT}$ is possible.

Theorem 8.6.1. *For the cost of $\mathcal{B}$ in the case of uniform matroids it holds that*

$$B \leq 2\,OPT + \frac{4\eta}{N-r},$$

where η is the total error of the predictor.

Proof. The proof can be found in [66]. It is a rather complicated proof involving the use of an intricate potential function. The proof is given in terms of the caching problem. In the following section we will establish the connection between the caching problem and the problem of maintaining a base of a uniform matroid with uniform costs where at most one element dies at each timestep. It remains an open question whether the simple proof we presented can be modified to prove this bound as well. □

Combining Theorem 8.5.1 and Theorem 8.6.1 we have the following theorem:

Theorem 8.6.2. *In the case of uniform matroids algorithm $\mathcal{B}$ pays at most:*

$$\min\left(OPT + 2\eta,\ 2OPT + \frac{4\eta}{N-r}\right).$$

As we will see in the next section the above result is tight up to constant factors.

8.7 Caching

The above analysis can be applied to the problem of learning-augmented caching yielding a surprisingly simple proof. The results are identical to those of [66] but we avoid the tedious case analysis and the use of complicated potential functions.

The k-caching problem consists of a fast memory containing k-pages and a slow memory containing N pages. At each timestep a page is requested. If the page is not in the fast memory (cache miss), we have to evict a page from the cache to make space for the requested page. In this case, the algorithm incurs a cost of 1. If the requested page is already present in the cache (cache hit) the algorithm incurs no cost. The goal is to achieve the least possible cost.

We will now argue that k-caching is a complementary problem to MMM for uniform matroids and actually when we solve a k-caching instance we simultaneously solve an instance of MMM for uniform matroids where the rank of the underlying matroid is N-k.

Suppose we have an instance of the MMM problem for uniform matroids with N elements/pages alive at each timestep (see the online approximation in section 2) and that at most one element dies at each timestep.

We partition the elements/pages into two distinct sets S and S'. Set S (the cache set) contains k elements/pages, while set S' (the base set) contains the rest $N-k$ of the elements/pages. Terms page and element can be used interchangeably in this setting. We will use the term element.

When a page request occurs then an element from the base set S' moves to the cache set S. The element that we evict from the cache, to make space for the incoming element, is moved to the base set S'. From the perspective of k-caching, we pay 1 for the cache miss and we evict a page to make space for the requested page. On the other hand, from the perspective of the MMM problem an element has died and we have to buy a new element so as to form a base. A subtle difference between the two settings is that the requested element at timestep t has to stay in the cache until timestep t+1 and thus cannot be acquired by the MMM algorithm until timestep t+1.

Furthermore, the optimal algorithm for k-caching is to evict the page that will be requested furthest in the future (Belady's rule). From the MMM perspective the optimal algorithm is to buy the element dying furthest in the future. Thus, the two algorithms select the same element. Finally, observe that the two algorithms have the same costs since when the one evicts a page and pays 1, the other one buys the element and also pays 1.

It is now clear that when we solve an instance of k-caching we simultaneously solve an instance of (N-k)-MMM where the underlying matroid is uniform and the acquisition cost for every element is 1.

Replacing $N-r$ with k in the results of the previous section yields the optimal bounds proved in [66].

Theorem 8.7.1. *For the learning-augmented caching problem, algorithm $\mathcal{B}$ pays at most:*

$$\min\left(OPT+2\eta,\ 2OPT+\frac{4\eta}{k}\right).$$

Moreover, we state the following lower bound, which is proved in [66] (Theorem 1.4):

Theorem 8.7.2. *The competitive ratio bound of any deterministic learning augmented online caching algorithm must be at least*

$$1+\Omega\left(\frac{1}{k}\frac{\eta}{OPT}\right).$$

This theorem also implies a lower bound for the case of uniform matroids we examined in the previous section.

8.8 Generalization

In this section we will remove the assumption that at most one element dies at each timestep. We will show that our results still hold when multiple elements die at each timestep.

In this scenario we have to consider the extended versions of $\mathcal{G}$ and $\mathcal{B}$. That is, we buy the elements that will die (or are predicted to die) furthest in the future such that our set remains independent. We continue this process until we have formed a base. Basically, we run the famous Kruskal's algorithm either with the true death times or with the predicted death times.

We will now extend the proof of theorem 8.5.1. Suppose that k elements die at timestep t. We have to buy k new elements to form a base. Suppose algorithms $\mathcal{G}$ and $\mathcal{B}$ choose $j \leq k$ different elements in order to form their bases. We add the $k - j$ common elements in the bases and consider those j different elements. Using theorem 8.2.3 we have that

$$A_t \leq A_{t-1} + j$$

We will prove that at timestep t we had at least j inversions, i.e. $M_t \geq j$.

Let us consider an element e_i that $\mathcal{B}$ bought, but $\mathcal{G}$ did not buy. If we add element e_i to the base of $\mathcal{G}$, a cycle is formed. We consider only the newly added elements of the cycle. We claim that one newly added element of $\mathcal{G}$ in the cycle forms an inversion with element e_i. Indeed, the true death time t_i of element e_i has to be less than all the respective times of the newly added elements of $\mathcal{G}$. Otherwise, algorithm $\mathcal{G}$ would have chosen e_i. Furthermore, the predicted death time t'_i of e_i has to be greater than that of the newly added elements otherwise algorithm $\mathcal{B}$ would have chosen one of these elements. Thus, element e_i forms an inversion with at least one of the newly added elements of $\mathcal{G}$.

This is true for all elements and thus we have $M_t \leq j$.

Thus, it holds that

$$A_t \leq A_{t-1} + M_t.$$

Summing over all timesteps $t = 1, \ldots, T$ we have that

$$B \leq G + M.$$

Finally, we note that in the case of elements with non-uniform acquisition costs it has been proven that our predictions cannot aid us to go beyond the classical online algorithms even in the case of uniform matroids [9].

8.9 Conclusions

In this chapter we extended the learning-augmented framework we used in chapter 4, to solve the paging problem, for the more general case of the MMM problem. Since the optimal algorithm for the MMM problem is not a simple greedy algorithm as is the case for the paging problem we had to modify our general approach. In the previous chapter of this thesis, we argued that in such cases one might rethink what the predictor should predict. In this chapter, however, we chose a different approach. The predictor remained the same and we tried to mimic the decisions of the greedy offline algorithm which is a good approximation of the optimal algorithm. The greedy algorithm based its decisions on the death times of the elements and thus we identified that this was these were the predictions we needed.

The basic idea is that since we have an offline greedy algorithm that is a good approximation of the optimal algorithm we will try to use the predictions in order to be as good as the greedy algorithm. As a result, our algorithm will also be a good approximation of the optimal algorithm.

The next crucial idea was the trick with the (meta-)algorithm A_t, which was also used in the analysis of the learning-augmented algorithms for the MTS problem in the previous chapter. By comparing algorithm A_t to algorithm A_{t-1} we were able to isolate one timestep and concentrate our analysis to what happens in this specific timestep. Using the equivalence of the MMM and MSM problems we were able to bound the difference of these two algorithms using the error η.

The question of whether our bounds are tight remains open for the general case. It would be interesting to provide such bounds for different types of matroids such as the graphical matroid. Finally, an important result would be to prove that algorithm $\mathcal{G}$ is a constant approximation of the optimal algorithm in the case of uniform costs.

Βιβλιογραφία

[1] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.

[2] Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[3] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms (TALG)*, 2(4):640–660, 2006.

[4] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM Journal on Computing*, 39(2):361–370, 2009.

[5] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc Renault. Online computation with untrusted advice. *arXiv preprint arXiv:1905.05655*, 2019.

[6] Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. Online bin packing with predictions. *arXiv preprint arXiv:2102.03311*, 2021.

[7] Spyros Angelopoulos and Pascal Schweitzer. Paging and list update under bijective analysis. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 1136–1145. SIAM, 2009.

[8] Barbara M Anthony and Anupam Gupta. Infrastructure leasing problems. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 424–438. Springer, 2007.

[9] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, pages 345–355. PMLR, 2020.

[10] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. *arXiv preprint arXiv:2006.01026*, 2020.

[11] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 542–547. IEEE, 1997.

[12] Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. *arXiv preprint arXiv:2010.11632*, 2020.

[13] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM (JACM)*, 59(4):1–24, 2012.

[14] Nikhil Bansal, Kedar Dhamdhere, Jochen Könemann, and Amitabh Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. *Algorithmica*, 40(4):305–318, 2004.

[15] Luca Becchetti and Stefano Leonardi. Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 94–103, 2001.

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

[17] Shai Ben-David, Allan Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

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

[19] Allan Borodin and Ran El-Yaniv. Online algorithms and competitive analysis, 1998.

[20] Allan Borodin, Nathan Linial, and Michael E Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM (JACM)*, 39(4):745–763, 1992.

[21] Peter Brucker. Scheduling algorithms. *Journal-Operational Research Society*, 50:774–774, 1999.

[22] Niv Buchbinder, Shahar Chen, Joshep Seffi Naor, and Ohad Shamir. Unified algorithms for online learning and competitive analysis. In *Conference on Learning Theory*, pages 5–1. JMLR Workshop and Conference Proceedings, 2012.

[23] Niv Buchbinder, Kamal Jain, and Joseph Seffi Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *European Symposium on Algorithms*, pages 253–264. Springer, 2007.

[24] Niv Buchbinder and Joseph Naor. Improved bounds for online routing and packing via a primal-dual approach. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 293–304. IEEE, 2006.

[25] Niv Buchbinder and Joseph Naor. *The design of competitive online algorithms via a primal-dual approach*. Now Publishers Inc, 2009.

[26] Bo Chen, Chris N Potts, and Gerhard J Woeginger. A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of combinatorial optimization*, pages 1493–1641, 1998.

[27] Jakub Chledowski, Adam Polak, Bartosz Szabucki, and Konrad Tomasz Żołna. Robust learning-augmented caching: An experimental study. In *International Conference on Machine Learning*, pages 1920–1930. PMLR, 2021.

[28] Christian Coester and Elias Koutsoupias. The online k-taxi problem. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 1136–1147, 2019.

[29] Thomas M Cover and David H Gluss. Empirical bayes stock market portfolios. *Advances in applied mathematics*, 7(2):170–181, 1986.

[30] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

[31] Amos Fiat, Richard M Karp, Michael Luby, Lyle A McGeoch, Daniel D Sleator, and Neal E Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[32] Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 48(3):410–428, 1994.

[33] Rudolf Fleischer. On the bahncard problem. *Theoretical Computer Science*, 268(1):161–174, 2001.

[34] Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive steiner tree online. *SIAM Journal on Computing*, 45(1):1–28, 2016.

[35] Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In *International Colloquium on Automata, Languages, and Programming*, pages 563–575. Springer, 2014.

[36] Monika R Henzinger and Valerie King. Maintaining minimum spanning trees in dynamic graphs. In *International Colloquium on Automata, Languages, and Programming*, pages 594–604. Springer, 1997.

[37] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, pages 285–294, 2021.

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

[39] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):325–346, 2003.

[40] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. *arXiv preprint arXiv:2006.09509*, 2020.

[41] Anna R Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgement and other stories about e/(e-1). In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 502–509, 2001.

[42] Anna R Karlin and Elias Koutsoupias. Beyond competitive analysis., 2020.

[43] Anna R Karlin, Mark S Manasse, Lyle A McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 301–309, 1990.

[44] Rohan Kodialam. Competitive algorithms for an online rent or buy problem with variable demand. *SIAM Undergraduate Research Online*, 7:233–245, 2014.

[45] Rohan Kodialam. Optimal algorithms for ski rental with soft machine-learned predictions. *arXiv preprint arXiv:1903.00092*, 2019.

[46] Elias Koutsoupias and Christos H Papadimitriou. Beyond competitive analysis. *SIAM Journal on Computing*, 30(1):300–317, 2000.

[47] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[48] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[49] Ravi Kumar, Manish Purohit, and Zoya Svitkina. Improving online algorithms via ml predictions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9684–9693, 2018.

[50] Thomas Lavastida, Benjamin Moseley, R Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. *arXiv preprint arXiv:2011.11743*, 2020.

[51] Thodoris Lykouris and Sergei Vassilvtiskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3296–3305. PMLR, 2018.

[52] Mark S Manasse, Lyle A McGeoch, and Daniel D Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11(2):208–230, 1990.

[53] Lyle A McGeoch and Daniel D Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(1):816–825, 1991.

[54] Adam Meyerson. The parking permit problem. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, pages 274–282. IEEE, 2005.

[55] Michael Mitzenmacher. A model for learned bloom filters, and optimizing by sandwiching. *arXiv preprint arXiv:1901.00902*, 2019.

[56] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. *arXiv preprint arXiv:1902.00732*, 2019.

[57] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. *arXiv preprint arXiv:2006.09123*, 2020.

[58] Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical computer science*, 130(1):17–47, 1994.

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

[60] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1845. SIAM, 2020.

[61] Tim Roughgarden. Beyond worst-case analysis. *Communications of the ACM*, 62(3):88–96, 2019.

[62] Tim Roughgarden. Resource augmentation., 2020.

[63] Alexander Schrijver. Combinatorial optimization polyhedra and efficiency volume b matroids, trees, stable sets chapters 39-69. *Algorithms and Combinatorics*, 24(1):ALL–ALL, 2003.

[64] Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[65] Gal Tamir and Hadas Shachnai. *Algorithms for Combinatorial Reoptimization.* PhD thesis, Computer Science Department, Technion, 2016.

[66] Alexander Wei. Better and simpler learning-augmented online caching. *arXiv preprint arXiv:2005.13716*, 2020.

[67] Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. *arXiv preprint arXiv:2010.11443*, 2020.

[68] Laurence A Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.