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

Αλγόριθμοι Καθοδηγούμενοι από Δεδομένα για Προβλήματα Συνεκτικότητας σε Γραφήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΛΤΙΑΔΗΣ ΣΤΟΥΡΑΣ

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

Αθήνα, Μάρτιος 2022

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

Αλγόριθμοι Καθοδηγούμενοι από Δεδομένα για Προβλήματα Συνεκτικότητας σε Γραφήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΙΛΤΙΑΔΗΣ ΣΤΟΥΡΑΣ

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4η Μαρτίου 2022.

Δημήτριος Φωτάκης Αναπληρωτής Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.	Χρήστος Τζάμος Αναπληρωτής Καθηγητής University of Wisconsin-Madison	Αριστείδης Παγουρτζής Καθηγητής ΣΗΜΜΥ Ε.Μ.Π.

Αθήνα, Μάρτιος 2022

Μιλτιάδης Στούρας

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

Περίληψη

Η παρούσα διπλωματική εργασία μελετά το πρόβλημα της συνεκτικότητας γραφημάτων υπό την σκοπιά του τομέα Data-Driven Algorithm Design. Η συνεκτικότητα γραφημάτων είναι ένα θεμελιώδες πρόβλημα συνδιαστικής βελτιστοποίησης που συναντάται αρκετά συχνά σε πρακτικά προβλήματα και πολλές φορές σε στοχαστικό περιβάλλον. Ένα τέτοιο παράδειγμα αποτελούν τα δίκτυα υπολογιστών τα οποία βασίζονται στην συνεκτικότητά τους για να λειτουργούν. Πολλές φορές, όμως, οι συνδέσεις μεταξύ υπολογιστών μπορεί να είναι αναξιόποιηστες (π.χ. να έχουν υψηλό θόρυβο) ή να έχουν βγει προσωρινά εκτός λειτουργίας. Μετά από κάποιο φυσικό συμβάν που μπορεί να θέσει εκτός λειτουργίας διάφορες συνδέσεις του δικτύου, θα θέλαμε να μπορούμε να βρούμε ένα συνδετικό δέντρο ώστε να επαναφέρουμε το δίκτυο σε λειτουργία. Ωστόσο, ο έλεγχος για την λειτουργικότητα συνδέσεων μπορεί να είναι αρκετά κοστοβόρος, επομένως το ζητούμενο είναι να ανακαλύψουμε ένα συνδετικό δέντρο κάνοντας όσο λιγότερους ελέγχους μπορούμε. Ακολουθώντας την γραμμή έρευνας που ξεκίνησε από τους Chawla et al. ([CGT+20]), ορίζουμε δύο κλάσεις αλγορίθμων με βάση την προσαρμοστικότητά τους και προσπαθούμε να προσεγγίσουμε το κόστος της βέλτιστης μη-προσαρμοστικής στρατηγικής για την κατανομή εισόδων που αντιμετωπίζουμε. Αποδεικνύουμε ότι είναι NP-hard να προσεγγίσουμε το κόστος της βέλτιστης μη-προσαρμοστικής στρατηγικής με λόγο προσέγγισης μικρότερο από λογαριθμικό χρησιμοποιώντας έναν υπολογιστικά αποδοτικό μη-προσαρμοστικό αλγόριθμο και σχεδιάζουμε προσαρμοστικούς αλγορίθμους που καταφέρνουν να προσεγγίσουν το κόστος αυτό με σταθερό λόγο προσέγγισης σε διάφορες οικογένειες γραφημάτων.

Λέξεις κλειδιά

Αξιοποίηση Δεδομένων, Αλγόριθμοι Καθοδηγούμενοι από Δεδομένα, Προσαρμοστικοί Αλγόριθμοι, Συνεκτικότητα Γραφημάτων, Τυχαία Γραφήματα, Treewidth

Abstract

This diploma thesis revisits the problem of graph connectivity under a data-driven perspective. Graph connectivity is a fundamental combinatorial optimization problem and there are many real world cases where it appears in a stochastic probing setting. A real-world example might be a computer network, or perhaps a gas pipeline, where the connections between some nodes have been damaged or are unreliable. After an event that has damaged an unknown subset of the network's links, we would like to make the network operational again by finding a spanning tree formed by non-damaged links. However, checking the reliability of the connections may be costly. For example in the case of the gas pipeline, we would have to physically visit the link's location and perform diagnostic tests. As a result, we would like to discover a spanning tree by checking as few links as possible. Following the work of [CGT+20], we define two natural classes of algorithms based on their adaptivity level and try to compete with the optimal non-adaptive cost for the input distribution. We show that it is NP-hard to approximate the optimal non-adaptive strategy within a sublogarithmic factor and develop adaptive strategies that achieve a constant competitive ratio with respect to the optimal non-adaptive strategy in certain graph families.

Key words

Data-Driven Algorithms, Adaptive Algorithms, Graph Connectivity, Random Graphs, Treewidth

Ευχαριστίες

Η δουλειά της παρούσας διπλωματικής έχει γίνει από κοινού με τον κ. Φωτάκη, τον Χρήστο Τζάμο, την Ευαγγελία Γεργατσούλη και τον Χάρη Πίπη. Θέλω να ευχαριστήσω πολύ τον κ. Φωτάκη για την εμπιστοσύνη που μου έδειξε από νωρίς, για την όμορφη συνεργασία μας και την πολύτιμη καθοδήγησή του. Θέλω επίσης να ευχαριστήσω τον Χρήστο για την συνεργασία μας και τον χρόνο που αφιέρωσε στα πρώτα μας ερευνητικά βήματα. Ακόμα, θέλω να ευχαριστήσω θερμά την Ευαγγελία για όλη την βοήθειά της και τις πολύτιμες συμβουλές σε προσωπικό και ακαδημαϊκό επίπεδο. Τέλος, θέλω να ευχαριστήσω πολύ την οικογένειά μου, την μητέρα μου και την αδερφή μου, για την υπομονή και την στήριξή τους.

Κλείνοντας, θέλω να ευχαριστήσω θερμά όλους τους κοντινούς μου φίλους. Οι πολύωρες συζητήσεις μας με διαμόρφωσαν σαν άνθρωπο, ενώ οι όμορφες βόλτες και οι στιγμές που περάσαμε μαζί έκαναν τα φοιτητικά μου χρόνια αξέχαστα. Ερχόμενος στην σχολή πριν από πέντε χρόνια δεν θα μπορούσα να έχω φανταστεί αυτό το υπέροχο ταξίδι, όπως και τώρα δεν μπορώ να φανταστώ τις όμορφες στιγμές που επιφυλάσσει το μέλλον. Νιώθω πραγματικά ευγνώμων για όσα έζησα ως φοιτητής και ανυπομονώ, πλέον, για το επόμενο κεφάλαιο.

Μιλτιάδης Στούρας,

Αθήνα, 4η Μαρτίου 2022

Περιεχόμενα

Περίληψη 5

Abstract 7

Ευχαριστίες 9

Περιεχόμενα 11

Εκτεταμένη Ελληνική Περίληψη 13

Κείμενο στα αγγλικά 29

1. Introduction 29
1.1 Contributions 30
1.2 Organization 31

2. Data-Driven Algorithm Design 33
2.1 Beyond the Worst Case Analysis of Algorithms 33
2.1.1 The Benefits of Worst-Case Analysis 33
2.1.2 Goals of the Analysis of Algorithms 34
2.1.3 The Cons of Worst-Case Analysis 34
2.2 Data-Driven Approaches 35
2.2.1 Algorithm design as a distributional learning problem 35
2.2.2 The Pseudo-dimension of Algorithms 36
2.2.3 The pseudo-dimension of greedy heuristics 37
2.2.4 Clustering Problems 39
2.2.5 Other Applications 41
2.2.6 Open Directions 42

3. Pandora's Box Problem 43
3.1 Problem Definition 43
3.2 Natural classes of Algorithms based on Adaptivity 44
3.3 A reduction to scenario-aware strategies and its implications to learning 44
3.3.1 Ski Rental with varying buy costs 45

3.3.2 Learning a good probing order . . . 46
3.3.3 LP Formulations of SPA and NA strategies . . . 46
3.4 Competing with the Non-Adaptive Benchmark . . . 47
3.5 Extension to feasibility constraints . . . 48

4. Graph Connectivity . . . 51
4.1 Problem Definition . . . 51
4.2 Classes of Algorithms Based on Adaptivity . . . 52
4.2.1 Learning a good probing order . . . 52
4.2.2 LP Formulation of Non-Adaptive Strategies . . . 53
4.2.3 Finding the best Non-Adaptive Algorithm . . . 54
4.3 Deriving Hardness from the Oracle Setting with One-Sided Error . . . 56
4.3.1 Definition and reduction to the Basic Model . . . 56
4.3.2 Formulation as a Markov Decision Process . . . 58
4.4 Competing with the Non-Adaptive Benchmark . . . 60
4.4.1 Separating Cut Selection and Querying a Cut . . . 60
4.4.2 Warm-Up Result: Series Parallel Graphs . . . 63
4.4.3 An Algorithm Based on the Moldgraph's Treewidth . . . 65
4.4.4 An Algorithm for Sparse and Minor-Closed Graphs . . . 69
4.5 Concluding Remarks & Future Work . . . 70

Bibliography . . . 73

Appendix 79

A. Tree Decompositions . . . 79
A.1 Definition of Treewidth . . . 79
A.2 Calculating the Treewidth of a graph . . . 82
A.3 "Nice" Tree Decompositions . . . 83

B. Ommited Proofs . . . 85

Εκτεταμένη Ελληνική Περίληψη

Το βασικό σκέλος της παρούσας διπλωματικής εργασίας έχει αποδωθεί στην αγγλική γλώσσα. Σε αυτό το κομμάτι, 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) περιεχόμενο της διπλωματικής εργασίας.

Σχεδιασμός Αλγορίθμων με την Χρήση Δεδομένων

Το Data-Driven Algorithm Design είναι μια νέα κατεύθυνση έρευνας που υπάγεται στον τομέα "Beyond Worst Case Analysis of Algorithms". Στον τομέα αυτό, οι ερευνητές προσπαθούν να ορίσουν μαθηματικά μοντέλα που να εξηγούν εμπειρικά φαινόμενα σχετικά με την απόδοση αλγορίθμων. Ωστόσο, η έρευνα έχει και μία δεύτερη διάσταση στην οποία οι στόχοι είναι η θεωρία να δίνει χρήσιμες συμβουλές για το ποιόν αλγόριθμο πρέπει να χρησιμοποιήσει κάποιος για ένα συγκεκριμένο πρόβλημα και γενικότερα πως να σχεδιάζει αλγορίθμους που θα δουλεύουν ικανοποιητικά για συγκεκριμένες εισόδους.

Τα Πλεονεκτήματα της Ανάλυσης Χειρότερης Περίπτωσης

Η ανάλυση χειρότερης περίπτωσης είναι μία σχεδιαστική επιλογή στην ανάλυση αλγορίθμων, κατά την οποία το προφίλ απόδοσης ενός αλγορίθμου συνοψίζεται από την χειρότερη επίδοσή του σε οποιαδήποτε είσοδο ενός συγκεκριμένου μεγέθους. Αν και άκομψη, η ανάλυση χειρότερης περίπτωσης έχει αρκετά πλεονεκτήματα και για αρκετούς λόγους παραμένει η κυρίαρχη επιλογή για ανάλυση αλγορίθμων στην θεωρητική πληροφορική.

i) Μία καλή επίδοση χειρότερης περίπτωσης είναι το ισχυρότερο αποτέλεσμα που μπορεί να αποδείξει κανείς για έναν αλγόριθμο. Πιστοποιεί ότι ο αλγόριθμος τα πηγαίνει καλά σε όλες τις εισόδους και οι χρήστες δεν χρειάζεται να κατανοούν αν οι είσοδοι που θέλουν να αντιμετωπίσουν ανήκουν στις εισόδους στις οποίες ο αλγόριθμος συμπεριφέρεται καλά. Ως αποτέλεσμα, αυτού του είδους η ανάλυση είναι πολύ χρήσιμη για αλγορίθμους που πρόκειται να χρησιμοποιηθούν σε πολλά διαφορετικά περιβάλλοντα, όπως για παράδειγμα ο αλγόριθμος ταξινόμησης μιας προγραμματιστικής βιβλιοθήκης.

ii) Η επίδοση του αλγορίθμου στην χειρότερη περίπτωση είναι πολλές φορές αρκετά ευκολότερο να υπολογιστεί από ότι άλλες εναλλακτικές όπως η μέση επίδοση του αλγορίθμου για μια κατανομή εισόδων.

iii) Για ένα μεγάλο αριθμό θεμελιωδών προβλημάτων υπάρχουν ήδη αλγόριθμοι με καλή επίδοση χειρότερης περίπτωσης, κάτι που χρίζει την αναζήτηση εναλλακτικής αχρείαστη.

Οι Στόχοι της Ανάλυσης Αλγορίθμων

Πριν σπεύσουμε να κατακρίνουμε την ανάλυση χειρότερης περίπτωσης, πρέπει να κάνουμε ένα βήμα πίσω και να σκεφτούμε ποιοί ακριβώς είναι οι στόχοι μας από την ανάλυση αλγορίθμων. Ο T. Roughgarden στο [Rou21] αναφέρει τρεις τέτοιους στόχους:

i) **Πρόβλεψη επίδοσης**. Ο πρώτος στόχος μας είναι να μπορούμε να προβλέψουμε την πρακτική απόδοση αλγορίθμων. Σε κάποιες περιπτώσεις, παίρνοντας ως δεδομένο ένα εμπειρικό φαινόμενο (π.χ. ότι η LRU πολιτική στις caches αποδίδει καλά), ψάχνουμε ένα μαθηματικό μοντέλο που να εξηγεί αυτό το φαινόμενο. Σε άλλες περιπτώσεις, μπορεί να αναζητούμε μια θεωρία που να μπορεί να μας συμβουλέψει ορθά για το αν ένας αλγόριθμος θα αποδόσει καλά στην εφαρμογή που τον χρειαζόμαστε ή όχι.

ii) **Εντοπισμός βέλτιστων αλγορίθμων**. Ο δεύτερος στόχος είναι να μπορούμε να ταξινομούμε αλγορίθμους με βάση την απόδοσή τους και ιδανικά να μπορούμε να εντοπίσουμε τον καλύτερο από αυτούς. Σε κάθε περίπτωση, δεδομένων δύο αλγορίθμων A και B, μια μέθοδος ανάλυσης θα πρέπει να μπορεί να εκφέρει μια άποψη για το ποιός από τους δύο είναι καλύτερος.

iii) **Σχεδιασμός νέων αλγορίθμων**. Τέλος, θέλουμε ένα καλώς ορισμένο πλαίσιο ανάλυσης αλγορίθμων ώστε να καθίσταται σαφές ποιά είναι η κατεύθυνση βελτιστοποίησης ώστε να μπορούμε να σκεφτούμε νέους αποδοτικούς αλγορίθμους.

Τα Μειονεκτήματα της Ανάλυσης Χειρότερης Περίπτωσης

Η ανάλυση χειρότερης περίπτωσης έχει λειτουργήσει πολύ καλά για πάρα πολλά προβλήματα συνδιαστικής βελτιστοποίησης. Ωστόσο, δεν πρέπει να αφήνουμε το γεγονός αυτό να μας παραπλανεί σχετικά με το εύρος της πραγματικής της χρησιμότητας.

i) **Υπερβολικά απαισιόδοξες προβλέψεις επίδοσης**. Όπως φαίνεται και από το όνομά της, η ανάλυση χειρότερης περίπτωσης δίνει μια αρκετά απαισιόδοξη εκτίμηση για την απόδοση των αλγορίθμων στην πράξη. Ένα πολύ γνωστό παράδειγμα είναι η μέθοδος Simplex για τον Γραμμικό Προγραμματισμό της οποίας ο χρόνος εκτέλεσης τυπικά αυξάνεται με ικανοποιητικό ρυθμό καθώς αυξάνεται το μέγεθος της εισόδου. Ωστόσο, η πολυπλοκότητά της είναι εκθετική στην χειρότερη περίπτωση, κάτι που έρχεται σε αντίθεση με την πρακτικότητά της.

ii) **Λανθασμένη ταξινόμηση αλγορίθμων**. Οι υπερβολικά απαισιόδοξες προβλέψεις μας δυσκολεύουν να αποφαθούμε για το ποιός αλγόριθμος είναι αυτός που πρέπει να διαλέξουμε για μια πρακτική εφαρμογή. Για παράδειγμα, η ανάλυση χειρότερης περίπτωσης δεν μπορεί να διακρίνει ποιά πολιτική από τις LRU και FIFO είναι καλύτερη, με την πρώτη να έχει σημαντικά καλύτερη απόδοση στην πράξη.

iii) **Απουσία μοντέλου δεδομένων**. Η ανάλυση χειρότερης περίπτωσης έχει ένα έμμεσο μοντέλο δεδομένων, στο οποίο η είσοδος που επιλέγεται είναι επίτηδες επιλεγμένη για να μην αποδίδει καλά ο αλγόριθμος. Πέρα από εφαρμογές ασφάλειας, αυτή η προσέγγιση είναι αρκετά παράλογη τόσο για την ανάλυση όσο και για τον σχεδιασμό αλγορίθμων. Εξ' άλλου, οι ιδιότητες των εισόδων που θα αντιμετωπίσουμε είναι εκείνες που ξεχωρίζουν τους πραγματικά καλούς αλγορίθμους. Για παράδειγμα, στην προηγούμενη περίπτωση η LRU είναι πρακτικά καλύτερη επειδή τα αιτήματα εμφανίζουν "locality of reference", το οποίο σημαίνει ότι οι πιο πρόσφατα χρησιμοποιημένες σελίδες είναι πιο πιθανό να ζητηθούν ξανά.

Τα μειονεκτήματα αυτά καταδεικνύουν την ανάγκη για εναλλακτικούς τρόπου ανάλυσης αλγορίθμων που θα μπορούν να αποδεικνύουν ιδιότητες της απόδοσης αλγορίθμων μόνο σε συγκεκριμένες εισόδους που μας ενδιαφέρουν.

Ο Σχεδιασμός Αλγορίθμων σαν ένα πρόβλημα μάθησης κατανομών

Στα προβλήματα που συναντάει κανείς στην πράξη, αντί να χρησιμοποιήσει έτοιμους αλγορίθμους που έχουν κακή επίδοση στην χειρότερη περίπτωση, συνήθως ακολουθείται μία data-driven προσέγγιση: οι προγραμματιστές χρησιμοποιούν παλιά δεδομένα του προβλήματος που θέλουν να λύσουν και προσπαθούν να βρουν ποιός από τους αλγορίθμους που έχουν στην διάθεσή τους τα πηγαίνει καλύτερα στο πεδίο τους. Η προσέγγιση αυτή έχει χρησιμοποιηθεί στην πράξη σε πολλούς τομείς όπως η θεωρητική πληροφορική ([Fin98, ACCL06]), η τεχνητή νοημοσύνη ([HRG+13, XHHLB08]), η υπολογιστική βιολογία ([DK18]) και ο σχεδιασμός δημοπρασιών ([San03]). Ωστόσο, οι προσεγγίσεις αυτές, μέχρι στιγμής, χρησιμοποιούνταν χωρίς να υπάρχει κάποια θεωρητική βεβαίωση για την απόδοσή τους.

Οι T. Roughgarden και R. Gupta ([GR15]), πρότειναν το πρόβλημα αυτό να μοντελοποιηθεί σαν ένα πρόβλημα μάθησης κατανομών, χρησιμοποιώντας και επεκτείνοντας το PAC Learning ([Val84]) και το Statistical Learning Theory ([Vap99]). Συγκεκριμένα, το "application-specific" κομμάτι μοντελοποιείται ως μια άγνωστη κατανομή πάνω σε όλες τις πιθανές εισόδους του προβλήματος από την οποία οι αλγόριθμοι μπορούν να λαμβάνουν δείγματα. Ο στόχος των ερευνητικών προσπαθειών είναι να αποδείξουν ότι κάποιες μέθοδοι επιλογής αλγορίθμων δουλεύουν βέλτιστα για όλες τις πιθανές κατανομές εισόδων, αποδεικνύοντας οτι ένας συγκεκριμένος αριθμός δειγμάτων αρκεί για να είμαστε σίγουροι ότι ο αλγόριθμος που είναι ο καλύτερος στα δείγματα θα είναι σίγουρα και ο καλύτερος για την άγνωστη κατανομή εισόδων.

Μοντελοποίηση του προβλήματος από το [Bal21]. Έστω ένα αλγοριθμικό πρόβλημα (π.χ. ένα πρόβλημα clustering ή ένα πρόβλημα επιλογής υποσυνόλων). Συμβολίζουμε με Π το σύνολο των εισόδων του προβλήματος οι οποίες μας ενδιαφέρουν. Έστω ακόμα μια (πιθανώς άπειρη) συλλογή $\mathcal{A}$ από αλγορίθμους για αυτό το πρόβλημα, η οποία παραμετροποιείται από ένα σύνολο $\mathcal{P} \subseteq \mathbb{R}^d$. Συμβολίζουμε με A_ρ τον αλγόριθμο της συλλογής $\mathcal{A}$ που αντιστοιχεί στην παράμετρο ρ. Ακόμα, υποθέτουμε ότι μας δίνεται μια συνάρτηση $u : \Pi \times \mathcal{P} \to [0, H]$, όπου το $u(x, \rho)$ είναι ένα μέτρο της επίδοσης του αλγορίθμου A_ρ όταν του δοθεί η είσοδος $x \in \Pi$. Συμβολίζουμε με $u_\rho(\cdot)$ την συνάρτηση $u_\rho : \Pi \to [0, H]$ που επάγεται από τον αλγόριθμο A_ρ, όπου $u_\rho(x) = u(x, \rho)$. Παρατηρήστε πως η u είναι άνω και κάτω φραγμένη. Για παράδειγμα, στις περιπτώσεις

που η u μπορεί να είναι ο χρόνος εκτέλεσης ενός αλγορίθμου, το άνω φράγμα H μπορεί να είναι το ανώτατο επιτρεπτό χρονικό όριο εκτέλεσης των αλγορίθμων.

Το "application-specific" κομμάτι του προβλήματος μοντελοποιείται από μια άγνωστη κατανομή $\mathcal{D}$. Η μέθοδος που επιλέγει τον βέλτιστο αλγόριθμο για την κατανομή, λαμβάνει m ανεξάρτητα και ισόνομα δείγματα $x_1, \ldots, x_m \in \Pi$ από την κατανομή $\mathcal{D}$, και (πιθανώς έμμεσα) την επίδοση $u_\rho(x)$ κάθε αλγόριθμου $A_\rho \in \mathcal{A}$ σε κάθε είσοδο x_i. Η μέθοδος αυτή, χρησιμοποιεί τα δείγματα για να προτείνει έναν αλγόριθμο $A_\rho \in \mathcal{A}$ ο οποίος θα χρησιμοποιηθεί για μελλοντικές εισόδους που θα έρθουν επίσης από την κατανομή $\mathcal{D}$. Αναζητούμε μια μέθοδο επιλογής αλγορίθμων η οποία πάντα να επιλέγει έναν αλγόριθμο από την συλλογή $\mathcal{A}$ που να αποδίδει περίπου όσο καλά αποδίδει ο βέλτιστος αλγόριθμος A_{ρ^*} για την κατανομή $\mathcal{D}$ ο οποίος μεγιστοποιεί την αναμενόμενη τιμή $\mathbf{E}_{x\sim\mathcal{D}}\left[u_\rho(x)\right]$ πάνω σε όλους τους αλγορίθμους $A_\rho \in \mathcal{A}$.

Η Ψευδοδιάσταση Αλγορίθμων

Για να αποδείξουμε τα ζητούμενα αποτελέσματα, συνήθως χρησιμοποιείται το uniform convergence. Συγκεκριμένα, οι ερευνητές προσπαθούν να ποσοτικοποιήσουν το πόσα δείγματα χρειάζονται ώστε να είμαστε βέβαιοι με μεγάλη πιθανότητα ότι η μέση επίδοση ενός αλγορίθμου στα δείγματα της κατανομής να είναι αρκετά κοντά στην αναμενόμενη επίδοσή του στην κατανομή $\mathcal{D}$. Όπως γνωρίζουμε από την θεωρία μάθησης κατανομών, ο αριθμός των δειγμάτων εξαρτάται αποκλειστικά από ιδιότητες της κλάσης $\mathcal{A}$ των αλγορίθμων και συγκεκριμένα το πόσο εγγενώς διαφορετικές συμπεριφορές μπορεί να έχουν οι αλγόριθμοι της κλάσης. Η ψευδοδιάσταση μιας κλάσης αλγορίθμων αποτελεί μια ποσοτικοποίηση αυτών των χαρακτηριστικών.

Definition 0.0.1 (Ορισμός 2.1 του [Bal21], Ψευδοδιάσταση)**.** Έστω $\{u_\rho(\cdot)\}_\rho$ μια συλλογή συναρτήσεων επίδοσης που επάγονται από την συλλογή αλγορίθμων $\mathcal{A} = \{A_\rho\}_\rho$ και την συνάρτηση $u(x, \rho)$.

(a) Έστω ακόμα $\mathcal{S} = \{x_1, \ldots, x_m\} \subset \Pi$ ένα σύνολο εισόδων και $z_1, \ldots, z_m \in \mathbb{R}$ ένα σύνολο στόχων. Θα λέμε ότι τα $z_1, \ldots, z_m$ αποτελούν ένα κατακερματισμό του συνόλου $\mathcal{S}$ μέσω της $\{u_\rho(\cdot)\}_\rho$ αν και μόνο αν για κάθε υποσύνολο $T \subseteq \mathcal{S}$, υπάρχει κάποια παράμετρος $\rho \in \mathcal{P}$ τέτοια ώστε για κάθε στοιχείο $x_i \in T,\ u_\rho(x_i) \leq z_i$ και για κάθε $x_i \notin T,\ u_\rho(x_i) > z_i$. Θα λέμε ότι το $\mathcal{S}$ κατακερματίζεται από την συνάρτηση $\{u_\rho(\cdot)\}_\rho$ αν υπάρχουν $z_1, \ldots, z_m$ τα οποία να αποτελούν κατακερματισμό του μέσω της $\{u_\rho(\cdot)\}_\rho$.

(b) Έστω $\mathcal{S} \subseteq \Pi$ το μεγαλύτερο σύνολο που μπορεί να κατακερματιστεί από την $\{u_\rho(\cdot)\}_\rho$. Η *ψευδοδιάσταση* της συλλογής συναρτήσεων $\{u_\rho(\cdot)\}_\rho$ είναι $|\mathcal{S}|$.

Παρατηρείστε ότι όταν η συλλογή $\{u_\rho(\cdot)\}_\rho$ είναι ένα σύνολο δυαδικών συναρτήσεων, η ψευδοδιάσταση ανάγεται στην VC-διάσταση της κλάσης $\{u_\rho(\cdot)\}_\rho$.

Όπως και στην Θεωρία Μάθησης και την VC-διάσταση, για να αποδείξουμε την δειγματική πολυπλοκότητα κλάσεων αλγορίθμων στον data-driven τομέα, αρκεί να φράξουμε την ψευδοδιάσταση της κλάσης των αλγορίθμων. Το επόμενο θεώρημα διατυπώνει μαθηματικά την προηγούμενη θέση και αποτελεί μια επέκταση του Θεμελιώδους Θεωρήματος της Στατιστικής Θεωρίας Μάθησης.

Theorem 0.0.2 (Θεώρημα 2.2 του [Bal21]). *Έστω $d_{\mathcal{A}}$ η ψευδοδιάσταση της κλάσης των συναρτήσεων επίδοσης $\{u_\rho(\cdot)\}_\rho$ που επάγεται από την κλάση αλγορίθμων $\mathcal{A}$ και την συνάρτηση $u(x,\rho)$. Θεωρούμε ότι το σύνολο τιμών της $u(x,\rho)$ είναι ένα υποσύνολο του $[0,H]$. Τότε, για κάθε $\epsilon>0$, κάθε $\delta\in(0,1)$ και κάθε κατανομή $\mathcal{D}$ πάνω στο Π, $m=O\left(\frac{H^2}{\epsilon^2}(d_{\mathcal{A}}+\ln\frac{1}{\delta})\right)$ δείγματα αρκούν ώστε με πιθανότητα $1-\delta$ κατά το τράβηγμα m δειγμάτων $\mathcal{S}=\{x_1,\ldots,x_m\}\sim D^m$, να ισχύει ότι για κάθε $\rho\in\mathcal{P}$, η απόλυτη διαφορά της μέσης επίδοσης του αλγορίθμου A_ρ στο δείγμα και της αναμενόμενης επίδοσης του στην κατανομή $\mathcal{D}$ είναι μικρότερη από ϵ, δηλαδή*

$$\left|\frac{1}{m}\sum_{i=1}^{m}u_\rho(x_i)-\boldsymbol{E}_{x\sim\mathcal{D}}\left[u_\rho(x)\right]\right|\leq\epsilon$$

Το παραπάνω θεώρημα αποτελεί ένα θεμελιώδες αποτέλεσμα για τον σχεδιασμό αλγορίθμων χρησιμοποιώντας δείγματα και υποστηρίζει την πλειοψηφία των προσπαθειών του τομέα οι οποίες προσπαθούν με διάφορους τρόπους να δώσουν κάποια άνω φράγματα για την ψευδοδιάσταση της συλλογής αλγορίθμων με την οποία ασχολούνται. Στο κεφάλαιο 2 αναλύουμε κάποια βασικά αποτελέσματα του τομέα που χρησιμοποιούν την προαναφερθείσα μέθοδο.

Ανοιχτά Προβλήματα

Ο σχεδιασμός αλγορίθμων με την χρήση δειγμάτων πιθανώς να αλλάξει ριζικά τον τρόπο με τον οποίο αναλύουμε και σχεδιάζουμε αλγορίθμους για προβλήματα συνδιαστικής βελτιστοποίησης. Πέρα από την βελτίωση των ήδη υπάρχοντων μεθόδων καθώς και της εφαρμογής τους σε νέα προβλήματα, θα είχε ενδιαφέρον ο σχεδιασμός ενός νέου τρόπου ανάλυσης ο οποίος να οδηγούσε σε ακόμα καλύτερες και αυτοματοποιημένες μεθόδους επιλογής αλγορίθμων. Για παράδειγμα, θα ήταν ενδιαφέρον να εξερευνηθεί μια μέθοδος βασισμένη σε Reinforcement Learning, όπου θα μπορούσαμε να μαθαίνουμε state-based decision policies για τα προβλήματα που μας ενδιαφέρουν. Τέλος, θα ήταν ακόμα ενδιαφέρον να αναπτυχθούν εργαλεία που μας επιτρέπουν να μαθαίνουμε καθώς εξερευνούμε ένα instance, σε αντίθεση με το να μεταφέρουμε την αποκτηθείσα γνώση μας από πολλά instances.

Το Κουτί της Πανδώρας

Το πρόβλημα "Το Κουτί της Πανδώρας" διατυπώθηκε πρώτη φορά από τον Weitzman στο [Wei79] και από τότε έχει μια μεγάλη γραμμή έρευνας και γενικεύσεων ([CFG+02, GGM06, ASW13, GNS, GJSS19, CGT+20]). Το πρόβλημα αυτό μπορεί να περιγράψει πολλά προβλήματα βελτιστοποίησης με πιθανοτική είσοδο, όπου ο αλγόριθμος μπορεί να αποκτήσει πληροφορίες για κάποιες τυχαίες μεταβλητές πληρώνοντας κάποιο κόστος. Η αναζήτηση του βέλτιστου τρόπου να αποκτηθεί η απαραίτητη πληροφορία είναι ένα online decision-making πρόβλημα, καθώς κάθε κομμάτι πληροφορίας που αποκαλύπτεται στον αλγόριθμο επηρεάζει τις επόμενες αποφάσεις του.

Το Κουτί της Πανδώρας, είναι ένα πρόβλημα αναζήτησης όπου ψάχνουμε την φθηνότερη επιλογή μεταξύ αρκετών εναλλακτικών, ωστόσο δεν θέλουμε να σπαταλήσουμε αρκετό χρόνο ψάχνοντας. Ένα παράδειγμα τέτοιας περίπτωσης θα μπορούσε να είναι η αναζήτηση σπιτιού.

Σε αυτήν την περίπτωση έχουμε αρκετές επιλογές, όμως για να καταλάβουμε αν μας αρέσει ένα σπίτι και πως συγκρίνεται με τα άλλα σπίτια που έχουμε δει, θα πρέπει να κανονίσουμε μια συνάντηση και να το επισκεφθούμε. Ιδανικά θα θέλαμε να διαλέξουμε το σπίτι που καλύπτει καλύτερα τις ανάγκες μας χωρίς να σπαταλήσουμε αρκετό χρόνο κοιτώντας διαφορετικά σπίτια.

Πιό επίσημα, στο πρόβλημα αυτό δίνονται n κουτιά στους αλγορίθμους, κάθε ένα από τα οποία περιέχει μια άγνωστη στοχαστική επιβράβευση. Οι αλγόριθμοι μπορούν να ανοίγουν κουτιά με όποια σειρά θέλουν πληρώνοντας το κόστος κάθε κουτιού και αποφασίζουν αν θέλουν να συλλέξουν κάποια από τις επιβραβεύσεις που έχουν δει και να τερματίσουν ή να συνεχίσουν την αναζήτηση. Ο στόχος τους είναι να μεγιστοποιήσουν την συλλεγόμενη επιβράβευση μείον το συνολικό κόστος των κουτίων που άνοιξαν.

Μία σημαντική υπόθεση όλων των εργασιών πριν από το [CGT$^+$20] ήταν πως οι τυχαίες μεταβλητές που αντιστοιχούν στις κρυφές επιβραβεύσεις των κουτιών ήταν ανεξάρτητες. Ωστόσο, αυτή η υπόθεση δεν είναι ρεαλιστική. Για παράδειγμα, στην περίπτωση των σπιτιών, τα σπίτια μιας συγκεκριμένης περιοχής πιθανώς να έχουν ίδια χαρακτηριστικά και ως εκ τούτου η προσωπική μας εκτίμηση για αυτά θα είναι παρόμοια. Το [CGT$^+$20] είναι η πρώτη εργασία που μελετά το πρόβλημα αφήνοντας τις τυχαίες μεταβλητές να έχουν οποιαδήποτε εξάρτηση μεταξύ τους. Ακόμα, μελετούν το πρόβλημα υπό data-driven σκοπιά και είναι επίσης η πρώτη εργασία του τομέα στην οποία η κλάση αλγορίθμων πάνω στην οποία κάνουμε την βελτιστοποίηση δεν είναι παραμετροποιημένη.

Κλάσεις αλγορίθμων ανάλογα με την προσαρμοστικότητα

Η βέλτιστη λύση για την στοχαστική αναζήτηση θα είναι μια πλήρως προσαρμοστική στρατηγική η οποία επιλέγει το επόμενο κουτί που θα ανοίξει ανάλογα με όλα τα κόστη που έχει δει μέχρι στιγμής. Αν και αυτές είναι οι καλύτερες στρατηγικές για τις οποίες μπορεί να ελπίζει κάποιος, είναι αδύνατο να τις προσεγγίσουμε ή να τις μάθουμε από δείγματα. Για παράδειγμα, μπορεί τα κόστη μερικών πρώτων κουτιών να κωδικοποιούν την τοποθεσία ενός κουτιού με κόστος 0 ενώ κάθε άλλο κουτί πιθανώς να έχει άπειρο κόστος. Ενώ το βέλτιστο κόστος μπορεί να εντοπιστεί με λίγα ερωτήματα, οποιαδήποτε λογική προσέγγιση του βέλτιστου κόστους θα χρειαζόταν να μάθει ακριβώς την απεικόνιση από τα κόστη των πρώτων κουτιών στις πιθανές τοποθεσίες του βέλτιστου κόστους. Το να μάθουμε μια τέτοια απεικόνιση από δείγματα είναι αδύνατο μέσα από δείγματα. Για τον λόγο αυτό, οι συγγραφείς του [CGT$^+$20] όρισαν κάποιες πιο περιορισμένες κλάσεις αλγορίθμων με βάση την προσαρμοστικότητά τους, τις οποίες χρησιμοποιούν για να συγκρίνουν την επίδοση της στρατηγικής που σχεδιάζουν.

Οποιοσδήποτε αλγόριθμος για αυτό το πρόβλημα, περιγράφεται από ένα ζεύγος (σ, τ) όπου το σ είναι μια μεταθεση των κουτιών και αντιπροσωπεύει την σειρά με την οποία θα τα ανοίξει ο αλγόριθμος και το τ είναι ένας κανόνας τερματισμού - ο χρόνος στον οποίο ο αλγόριθμος σταματά να ανοίγει κουτιά και επιστρέφει το ελάχιστο κόστος που έχει δει μέχρι στιγμής. Όλοι οι αλγόριθμοι χωρίζονται στις κάτωθι κατηγορίες.

Definition 0.0.3 (Ορισμός 2.1 του [CGT$^+$20], Προσαρμοστικότητα Στρατηγικών)**.** Σε μία *Πλήρως-Προσαρμοστική (FA) στρατηγική*, τα σ και τ μπορεί να εξαρτώνται από κόστη που έχει δει ο αλγόριθμος μέχρι στιγμής, όπως περιγράφεται παραπάνω.

Σε μία *Μερικώς-Προσαρμοστική (PA) στρατηγική*, η ακολουθια σ είναι ανεξάρτητη από τα

κόστη που έχει δει ο αλγόριθμος ανοίγοντας κουτιά και έχει αποφασιστεί πριν ξεκινήσει το άνοιγμα κουτιών. Ωστόσο, ο κανόνας τερματισμού τ μπορεί να εξαρτάται από την είσοδο καθώς αυτή αποκαλύπτεται στον αλγόριθμο.

Σε μία *Μη-Προσαρμοστική (NA) στρατηγική*, και το σ και το τ είναι σταθερά και έχουν αποφασιστεί πριν αποκαλυφθεί οποιοδήποτε κόστος στον αλγόριθμο. Συγκεκριμένα, ο αλγόριθμος πάντα ανοίγει ένα συγκεκριμένο υποσύνολο κουτιών και επιστρέφει το ελάχιστο κόστος που παρατηρεί.

Οι συγγραφείς του [CGT$^+$20] μελετούν και δύο επεκτάσεις του προβλήματος στις οποίες οι αλγόριθμοι καλούνται να επιλέξουν περισσότερα από ένα κουτιά. Στην μία περίπτωση πρέπει να επιλέξουν k κουτιά και στην δεύτερη πρέπει να επιλέξουν επίσης k κουτιά τα οποία όμως να αποτελούν την βάση ενός Matroid. Η δεύτερη περίπτωση περιγράφει πολλά προβλήματα συνδιαστικής βελτιστοποίησης, συμπεριλαμβανομένου και του προβλήματος συνεκτικότητας γραφημάτων που μελετά οι παρούσα διπλωματική. Τα αποτελέσματα που αποδεικνύουν συνοψίζονται στον παρακάτω πίνακα και παρουσιάζονται αναλυτικότερα στο Κεφάλαιο 3.

	Single Option	k Options	Matroid of rank k
PA vs PA (Upper-bound)	9.22	$O(1)$	$O(\log k)$
FA vs NA (Lower-bound)	1.27	1.27	$\Omega(\log k)$

Συνεκτικότητα Γραφημάτων

Η συνεκτικότητα γραφημάτων είναι ένα θεμελιώδες πρόβλημα συνδιαστικής βελτιστοποίησης και απαντάται σε πολλά προβλήματα στην καθημερινότητα, σε αρκετά από τα οποία εμφανίζεται και σε πιθανοτικό περιβάλλον. Μία τέτοια εφαρμογή είναι τα δίκτυα υπολογιστών τα οποία βασίζονται στην συνεκτικότητά τους για να λειτουργήσουν. Πολλές φορές, όμως, οι συνδέσεις μεταξύ υπολογιστών μπορεί να είναι αναξιόπιστες (π.χ. να έχουν υψηλό θόρυβο) ή να έχουν βγει προσωρινά εκτός λειτουργίας. Μετά από κάποιο φυσικό συμβάν που μπορεί να θέσει εκτός λειτουργίας διάφορες συνδέσεις του δικτύου, θα θέλαμε να μπορούμε να βρούμε ένα συνδετικό δέντρο ώστε να επαναφέρουμε το δίκτυο σε λειτουργία. Ωστόσο, ο έλεγχος για την λειτουργικότητα συνδέσεων μπορεί να είναι αρκετά κοστοβόρος, επομένως το ζητούμενο είναι να ανακαλύψουμε ένα συνδετικό δέντρο κάνοντας όσο λιγότερους ελέγχους μπορούμε.

Στο παρελθόν έχει ερευνηθεί το η $s-t$ συνεκτικότητα ([FFX$^+$17]) και το ελάχιστο συνδετικό δέντρο ([GV04]) σε ένα παρόμοιο μοντέλο στο οποίο οι ερευνητές θεωρούσαν ότι τα γραφήματά τους είναι ER τυχαίοι γράφοι ([ER59]), κάνοντας την κρίσιμη υπόθεση ότι η πιθανότητα να έχει καταστραφεί μια ακμή είναι ανεξάρτητη από τις υπόλοιπες. Ωστόσο, αυτή η υπόθεση δεν είναι αρκετά ρεαλιστική. Στην εργασία μας, αφήνουμε τις πιθανότητες αυτές να έχουν οποιαδήποτε συσχέτιση και θεωρούμε ότι οι είσοδοι έρχονται από μια άγνωστη κατανομή πάνω στα συνεκτικά υπογραφήματα του γράφου που μας ενδιαφέρει.

Ορισμός Προβλήματος

Στο πρόβλημα αυτό, μας δίνεται ένα γράφημα G, το οποίο ονομάζουμε *γράφημα καλούπι*, και μια κατανομή $\mathcal{D}$ πάνω στα συνεκτικά υπογραφήματα του G. Η φύση δειγματολειπτεί ένα γρά-

φημα G' από την κατανομή $\mathcal{D}$, του οποίου το σύνολο ακμών είναι αρχικά κρυφό. Ονομάζουμε το G' *πραγματικό* ή *υποκείμενο* γράφημα.

Όλοι οι αλγόριθμοι έχουν την δυνατότητα να ρωτήσουν μια ακμή του G για να μάθουν αν αυτή υπάρχει (δηλαδή αν ανήκει στο σύνολο ακμών του G') ή οχι. Ο στόχος τους είναι να βρουν ένα συνδετικό δέντρο του υποκείμενου γραφήματος G', σπαταλώντας όσο το δυνατόν λιγότερο χρόνο συλλέγοντας πληροφορία. Υποθέτουμε ότι όλα τα πιθανά υποκείμενα γραφήματα G' είναι συνεκτικά, αφού η πλήρης ανακάλυψη της συνεκτικότητας ενός υποκείμενου γραφήματος του οποίου δεν γνωρίζουμε τον ακριβή αριθμό συνεκτικών συνιστωσών απαιτεί να ρωτήσουμε όλες τις ακμές του καλουπιού G.

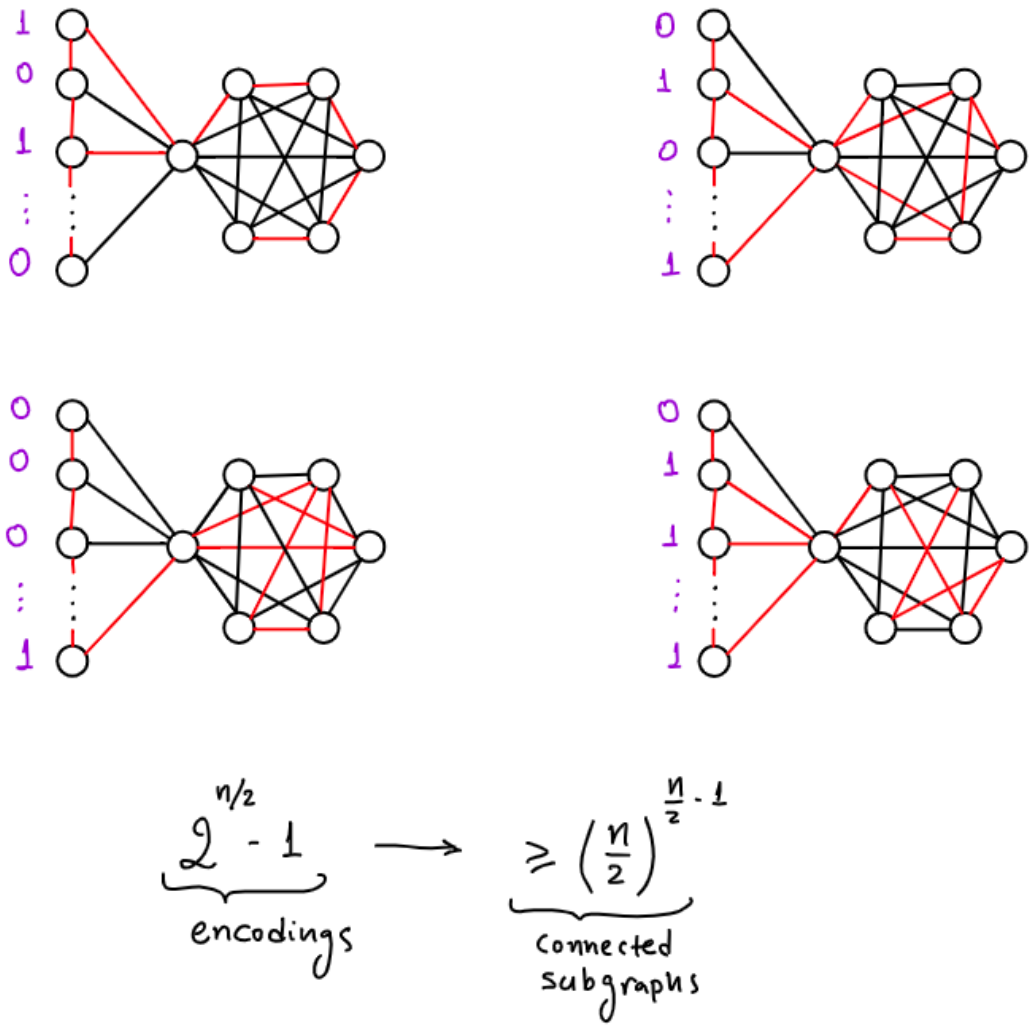
Πιο επίσημα, έστω $G' \sim \mathcal{D}$ το υποκείμενο υπογράφημα του G και έστω $Q_{G'}$ η τυχαία μεταβλητή που υποδηλώνει το σύνολο των ακμών που ρωτά ένας αλγόριθμος. Κάθε έγκυρο σύνολο $Q_{G'}$ πρέπει αναγκαστικά να περιέχει τουλάχιστον ένα συνεκτικό δέντρο του G. Ο στόχος μας είναι να ελαχιστοποιήσουμε τον αναμενόμενο αριθμό ερωτημάτων, ο οποίος είναι

$$\mathbf{E}_{G' \sim \mathcal{D}} \left[|Q_{G'}| \right]$$

Παρατηρείστε ότι το πρόβλημά μας ανάγεται στην Matroid επέκταση του Κουτιού της Πανδώρας, όπως αυτή ορίζεται στο [CGT+20], όταν δουλεύουμε με το Graphic Matroid. Το $\Omega(\log k)$ κάτω φράγμα στην προσεγγιστικότητα των Μη-Προσαρμοστικών στρατηγικών από οποιαδήποτε στρατηγική που αποδεικνύεται από τους Chawla et al. για το Partition Matroid, δεν μεταφέρεται στην δική μας περίπτωση. Θα ήταν ενδιαφέρον να διερευνηθεί αν υπάρχει κάποιο παρόμοιο κάτω φράγμα ή εάν η προσαρμοστικότητα μπορεί να μας δώσει καλύτερες προσεγγίσεις σε αυτήν την περίπτωση.

Κλάσεις αλγορίθμων με βάση την προσαρμοστικότητα

Όπως αναφέραμε και για το Κουτί της Πανδώρας, η βέλτιστη στρατηγική για το πρόβλημά μας θα ήταν ένας πλήρως προσαρμοστικός αλγόριθμος, ο οποίος αποφασίζει ποιά είναι η επόμενη ακμή που θα ρωτήσει με βάση τις απαντήσεις που έχει λάβει για αυτήν την είσοδο μέχρι στιγμής. Ωστόσο, αυτές οι στρατηγικές είναι αδύνατο να προσεγγιστούν με δείγματα από την κατανομή. Για παράδειγμα, μπορεί η ύπαρξη ή όχι κάποιων ακμών να κωδικοποιεί το υποκείμενο γράφημα. Δηλαδή, υπάρχει τρόπος να βρεθεί το κρυμμένο δέντρο με λίγα ερωτήματα αν γνωρίζουμε την αντιστοίχηση από τις κωδικοποιήσεις στα υποκείμενα γραφήματα. Ένα τέτοιο παράδειγμα φαίνεται στο παρακάτω σχήμα, όπου το γράφημα καλούπι είναι μια γραμμή $n/2$ κόμβων συνδεδεμένων με έναν κόμβο u ο οποίος συμμετέχει σε μια κλίκα με τους υπόλοιπους $n/2$ κόμβους. Σε κάθε υποκείμενο γράφημα οι ακμές της γραμμής δεν έχουν καταστραφεί και η ύπαρξη ή οχι των ακμών προς τον u αποτελεί μια κωδικοποίηση του υποκείμενου γραφήματος. Προφανώς για να μπορέσει μια στρατηγική να ανταγωνιστεί την βέλτιση για αυτήν την κατανομή, η οποία είναι μια στρατηγική που γνωρίζει και χρησιμοποιεί την αντιστοίχιση, θα πρέπει να μπορεί να διακρίνει όλες τις συναρτήσεις από τις $2^{n/2}$ κωδικοποιήσεις σε περισσότερα από $\left(\frac{n}{2}\right)^{n/2}$ δέντρα.



Επομένως, αν προσπαθούμε να ανταγωνιστούμε όλους τους πλήρως προσαρμοστικούς αλγορίθμους, ακόμα και εμπειρικά καλές στρατηγικές που μπορούν να γενικεύουν σε πλούσιες κατανομές θα έχουν κακή επίδοση συγκριτικά με όλους τους πιθανούς αλγορίθμους επειδή δεν μπορούν να ανταγωνιστούν τέτοιες στρατηγικές. Ως αποτέλεσμα, ορίζουμε τις παρακάτω κλάσεις αλγορίθμων που βασίζονται στην προσαρμοστικότητα των αλγορίθμων και χρησιμοποιούμε τον βέλτιστο μη-προσαρμοστικό αλγόριθμο ως ένα μέτρο σύγκρισης για να δούμε πόσο καλά αντιμετωπίζουμε την κατανομή.

Definition 0.0.4 (Προσαρμοστικότητα στρατηγικών)**.** Σε μία *Μη-Προσαρμοστική (NA)* στρατηγική, το σύνολο Q_G είναι σταθερό και αποφασίζεται εξ' αρχής. Οι αλγόριθμοι σε αυτήν την κλάση ρωτούν τις ακμές στο Q_G ανεξάρτητα από το υποκείμενο γράφημα και οποιαδήποτε πληροφορία έχουν λάβει από τις ερωτήσεις τους.

Μία *Πλήρως-Προσαρμοστική (FA)* στρατηγική αποφασίζει σε κάθε βήμα ποιά είναι η επόμενη ακμή που θα ρωτήσει βασισμένη σε πληροφορίες που έχει συλλέξει από τις μέχρι τώρα

ερωτήσεις καθώς και την γνώση της για την κατανομή. Με άλλα λόγια, το σύνολο Q_G δεν είναι αποφασισμένο εξ' αρχής, διαμορφώνεται δυναμικά και εξαρτάται από το υποκείμενο γράφημα.

Στρατηγικές που ανταγωνίζονται την βέλτιση Μη-Προσαρμοστική Στρατηγική

Αρχικά, αποδείξαμε (Λήμα 4.2.2) πως ένα πολυωνυμικά μεγάλο δείγμα από την κατανομή $\mathcal{D}$ αρκεί ώστε να είμαστε βέβαιοι ότι η μέση απόδοση κάθε μη-προσαρμοστικού αλγορίθμου θα είναι κοντά στην αναμενόμενη επίδοσή του στην κατανομή $\mathcal{D}$. Το αποτέλεσμα αυτό είναι αρκετά σημαντικό, καθώς μας επιτρέπει να αγνοήσουμε την κατανομή εισόδων $\mathcal{D}$, να επικεντρωθούμε μόνο σε ένα πολυωνυμικά μεγάλο δείγμα και να βρούμε την στρατηγική που είναι βέλτιστη για το δείγμα μας. Έχοντας αυτό το αποτέλεσμα, μπορούμε πλέον να περιγράψουμε τον βέλτιστο μη-προσαρμοστικό αλγόριθμο σαν ένα πρόβλημα Ακέραιου Προγραμματισμού:

$$\text{minimize} \quad \sum_{e \in E(G)} x_e \tag{LP-NA}$$

$$\text{subject to} \sum_{e \in \delta_i(A)} x_e \geq 1 \qquad \forall i \in [k], A \subset V(G)$$

Αποδεικνύουμε πως ένα randomized rounding του χαλαρωμένου Γραμμικού Προγράμματος μας δίνει μια μη-προσαρμοστική στρατηγική η οποία επιτυγχάνει μια $O(\log n + \log |\mathcal{D}|)$ προσέγγιση του βέλτιστου μη-προσαρμοστικού κόστους. Παράλληλα, με αναγωγή από το Set Cover, αποδεικνύουμε (Λήμμα 4.2.4) ότι είναι NP-hard να προσεγγίσουμε την βέλτιστη μη-προσαρμοστική στρατηγική με μια υπολογιστικά αποδοτική μη-προσαρμοστική στρατηγική, με λόγο προσέγγισης μικρότερο από $O(\log |\mathcal{D}|)$.

Έχοντας βρει λοιπόν την καλύτερη πιθανή μη-προσαρμοστική στρατηγική, έχει ενδιαφέρον πλέον να διερευνήσουμε εάν η προσαρμοστικότητα μπορεί να μας ρίξει κάτω από το φράγμα του $O(\log n)$. Στην προσπάθειά μας αυτή, ορίζουμε μια νέα θορυβώδη εκδοχή του προβλήματος η οποία αποτελεί bottleneck περίπτωση για το αρχικό πρόβλημα. Χρησιμοποιούμε την θορυβώδη εκδοχή του προβλήματος για να αποδείξουμε ένα σταθερό κάτω φράγμα για τον λόγο προσέγγισης καθώς και για να σχεδιάσουμε προσαρμοστικούς αλγόριθμους για το αρχικό πρόβλημα. Τελικά, διατυπώνουμε δύο στρατηγικές που επιτυγχάνουν σταθερό λόγο προσέγγισης, για γραφήματα με σταθερό treewidth και για αραιά και minor-closed γραφήματα.

Θορυβώδης εκδοχή του αρχικού προβλήματος

Στην Θορυβώδη εκδοχή του αρχικού προβλήματος, μας δίνεται ένα γράφημα καλούπι G και μια άγνωστη κατανομή $\mathcal{D}$ πάνω σε συνεκτικά υπογραφήματα του G. Η φύση τραβάει ένα δείγμα G' από την κατανομή $\mathcal{D}$. Οι αλγόριθμοι μπορούν να ρωτούν ένα μαντείο για το αν μια ακμή υπάρχει ή όχι. Το μαντείο λειτουργεί ως εξής: Αν η ερωτηθείσα ακμή υπάρχει τότε με πιθανότητα p απαντάει "Ναι" και με πιθανότητα $1 - p$ απαντάει "Όχι". Αν η ερωτηθείσα ακμή δεν υπάρχει τότε απαντάει "Όχι" με πιθανότητα 1. Με άλλα λόγια, αν πάρουμε μια θετική απάντηση είμαστε σίγουροι ότι η ακμή αυτή υπάρχει ενώ αν λάβουμε μια αρνητική απάντηση δεν

μπορούμε να ξέρουμε αν η ακμή δεν υπάρχει ή λάβαμε ψευδή πρόβλεψη. Ο στόχος μας είναι να ανακαλύψουμε ένα συνεκτικό δέντρο του υποκείμενου γραφήματος G' κάνοντας όσο λιγότερες ερωτήσεις μπορούμε.

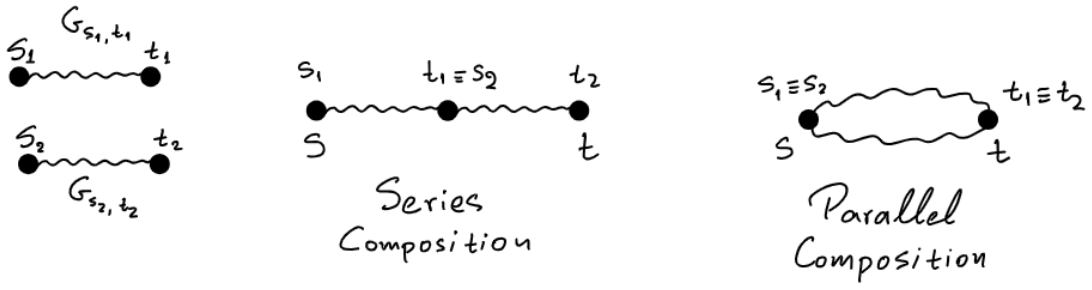
Αποδεικνύουμε (Θεώρημα 4.3.1) ότι το πρόβλημα αυτό ανάγεται στο αρχικό πρόβλημα και τα κάτω φράγματα για την απόδοση των στρατηγικών μεταφράζονται σε κάτω φράγματα για την προσεγγιστικότητα της βέλτιστης μη-προσαρμοστικής στρατηγικής από πλήρως προσαρμοστικούς αλγορίθμους. Ακόμα, δίνουμε έναν υπολογιστικό τρόπο εύρεσης κάτω φραγμάτων για το αρχικό πρόβλημα, περιγράφοντας την βέλτιστη στρατηγική της θορυβώδους εκδοχής σαν ένα Markov Decision Process και στο Θεώρημα 4.3.2 υπολογίζουμε ένα τέτοιο κάτω φράγμα.

Προσαρμοστικοί Αλγόριθμοι με βάση το Tree Decomposition

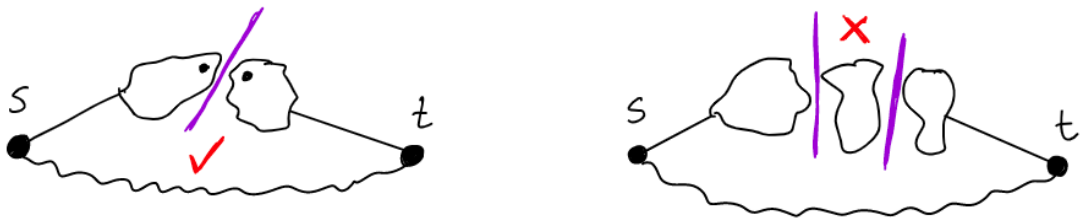
Θα περιοριστούμε αρχικά στα Series-Parallel γραφήματα τα οποία ορίζονται αναδρομικά ως εξής:

Definition 0.0.5 (Series-Parallel Graph)**.** Ένα γράφημα $G_{s,t}$ με μια πηγή s και μια καταβόθρα t είναι series-parallel (εφεξής SP) γράφημα αν και μόνο αν ισχύουν οι ακόλουθες συνθήκες

- Το $G_{s,t}$ αποτελείται μόνο από τις κορυφές s και t οι οποίες συνδέονται με μια ακμή
- Το $G_{s,t}$ μπορεί να δημιουργηθεί με την *παράλληλη κόλληση* δύο άλλων SP γραφημάτων G_{s_1,t_1} και G_{s_2,t_2}, δηλαδή ταυτίζοντας την κορυφή s_1 με την s_2 και την t_1 με την t_2.
- Το $G_{s,t}$ μπορεί να δημιουργηθεί με την *σειριακή κόλληση* δύο άλλων SP γραφημάωτν G_{s_1,t_1} και G_{s_2,t_2}, δηλαδή ταυτίζοντας την κορυφή t_1 με την s_2.



Μία πρώτη ιδέα θα ήταν να αξιοποιήσουμε τον αναδρομικό ορισμό του προβλήματος για να σχεδιάσουμε μια στρατηγική που δουλεύει αναδρομικά στους SP υπογράφους του καλουπιού και βρίκει ένα συνεκτικό δέντρο σε κάθε ένα από αυτά. Ωστόσο, η μόνη εγγύηση που έχουμε είναι ότι το προσκείμενο γράφημα θα είναι συνεκτικό, το οποίο δεν σημαίνει ότι κάθε υπογράφημα του καλουπιού θα συνδέεται με πραγματικές ακμές. Μπορεί, όπως φαίνεται παρακάτω, ένα από τα δύο υπογραφήματα να έχει ένα κόψιμο αλλά το γράφημα να παραμένει συνολικά συνεκτικό.



Μία σημαντική παρατήρηση όμως είναι ότι κάθε component δεν μπορεί να έχει πάνω από 2 s-t cuts, δηλαδή να χωρίζεται σε 3 συνεκτικές συνιστώσες, γιατί το συνολικό γράφημα δεν θα ήταν συνεκτικό. Εκμεταλλευόμενοι αυτήν την ιδιότητα σχεδιάζουμε τον παρακάτω αναδρομικό αλγόριθμο ο οποίος κάνει up-to-1 connected τα SP υπογραφήματα. Ο αλγόριθμος αυτός μας δίνει μια 4-προσέγγιση (βλ. Λήμμα 4.4.6).

Algorithm 1: SolveSP($G_{s,t}$): makes the SP Graph $G_{s,t}$ Up-To-1 Connected

if $V(G_{s,t}) == \{s,t\}$ **then**
  return $\emptyset$
end
Let $G_l,\ G_r$ be two SP components of $G_{s,t}$
$Q \leftarrow$ SolveSP$(G_l) \cup$ SolveSP(G_r)
if $G_{s,t}$ *is a Series Composition of* G_l *and* G_r **then**
  Let C_l, C_r be the uncovered $s-t$ cuts of G_l and G_r
  $S \leftarrow \{C_l, C_r\}$
  $Q \leftarrow Q \cup$ DISCOVER(S)
end
return Q

Ο παραπάνω αλγόριθμος γενικεύεται πολύ φυσικά για όλα τα γραφήματα μέσω του Tree Decomposition (βλ. Appendix A). Συγκεκριμένα, αρκεί να γίνει μια παρόμοια παρατήρηση (βλ. Λήμμα 4.4.7) πως αν δουλεύουμε αναδρομικά σε κάθε υποδέντρο του tree decomposition, τότε το επαγόμενο υπογράφημα των κορυφών που ανήκουν στο υποδέντρο μπορεί να σπάσει το πολύ σε k components, όπου k είναι το treewidth του γραφήματος καλουπιού. Χρησιμοποιώντας αυτήν την παρατήρηση διατυπώνουμε την παρακάτω στρατηγική η οποία αναδρομικά διατηρεί k components στο κάθε υποδέντρο του tree decomposition και επιτυγχάνει $O(k)$ λόγο προσέγγισης (βλ. Θεώρημα 4.4.8) που είναι σταθερός για τα γραφήματα με φραγμένο treewidth.

Προσαρμοστικοί Αλγόριθμοι για Sparse και Minor-Closed οικογένειες γραφημάτων

Τέλος, μελετάμε οικογένειες γραφημάτων ρ-αραιών και minor-closed, υπό τον παρακάτω ορισμό για αραιά γραφήματα.

Definition 0.0.6. Ένα γράφημα G με n κορυφές και m ακμές, ονομάζεται *ρ-αραιό* αν και μόνο αν $m \leq \rho n$.

Οι παραπάνω ιδιότητες φαίνονται αρκετά περιοριστικές, ωστόσο υπάρχουν ενδιαφέρουσες οικογένειες γραφημάτων που τις ικανοποιούν. Ένα παράδειγμα είναι τα επίπεδα γραφήματα που ικανοποιούν τον ορισμό για $\rho = 3$ και μάλιστα περιέχουν γραφήματα με unbounded treewidth.

Παρατηρούμε πως σε αυτά τα γραφήματα πάντα θα υπάρχει μια κορυφή με βαθμό το πολύ 2ρ, επομένως μια αποδοτική μέθοδος είναι να εντοπίζουμε την κορυφή ελαχίστου βαθμού και να την ενώνουμε με κάποιο γειτονικό (στο καλούπι) connected component. Ο αλγόριθμος αυτός περιγράφεται παρακάτω και επιτυγχάνει $O(\rho)$ λόγο προσέγγισης (βλ. Θεώρημα 4.4.11).

Algorithm 2: SolveSparse(G): Solves the problem in ρ-sparse minor-closed graphs.

```
if G is a single node then
    return ∅
end
Let u be the vertex with minimum degree in G.
e = DISCOVER(N(u))
Let G' be the resulting graph after contracting edge e.
return SolveSparse(G') ∪ {e}
```

Νέες Κατευθύνσεις

Κλείνοντας, θα ήταν ενδιαφέρον να διερευνηθεί αν μπορεί ο λόγος προσέγγισης $O(k)$ του treewidth αλγορίθμου στην θορηβώδη εκδοχή του προβλήματος να βελτιωθεί σε $O(\log k)$ για να είναι συνεπές με τον brute-force αλγόριθμο και να αναζητηθούν lower bounds παραμετροποιημένα με το treewidth του καλουπιού. Εκτός αυτού, θα είχε ενδιαφέρον μια διαφορετική ανάλυση των αποτελεσμάτων που θα λάμβανε υπόψιν της διάφορα χαρακτηριστικά της κατανομής εισόδου (π.χ. την εντροπία της) και θα έδινε μια πιο λεπτομερή εικόνα για τον λόγο προσέγγισης των αλγορίθμων. Άλλωστε, η δυσκολία του προβλήματος φαίνεται να είναι δισδιάστατη με την μια διάσταση να είναι η κατανομή εισόδου και την άλλη να είναι το γράφημα καλούπι. Συγκεκριμένα, αν κρατήσουμε σταθερό το γράφημα καλούπι τότε το πρόβλημα φαίνεται να γίνεται δυσκολότερο καθώς αυξάνεται η εντροπία της κατανομής εισόδου, ενώ αν περιοριστούμε για παράδειγμα στις uniform κατανομές, η δυσκολία του προβλήματος φαίνεται να αυξάνεται με το treewidth του καλουπιού. Τέλος, θα είχε ενδιαφέρον να αξιοποιηθεί η περιγραφή της θορυβώδους εκδοχής του προβλήματος σαν Markov Decision Process και να διευρευνηθεί το αν το Reinforcement Learning μπορεί να μας βοηθήσει να σχεδιάσουμε καλύτερες προσαρμοστικές στρατηγικές.

Κείμενο στα αγγλικά

Chapter 1

Introduction

From planning the fastest route to visit a number of locations in a given area, to navigating a search space to find the best solution, we are constantly required to solve large instances of problems that are provably hard to solve. Nowadays, we see large instances of *NP-hard* problems being solved efficiently by empirical solvers and heuristics. The most characteristic example is Integer Programming, for which a large number of empirical algorithms have been implemented and are widely being used by practitioners. It is a well established fact ([XHHLB08]) that each empirical solver has its own region of dominance in the space of all possible inputs. This means that if we fix an Integer Programming instance and vary over possible solvers, we will observe running times that differ by orders of magnitude. Therefore, there are no "silver bullet" off-the-self algorithms that work in all applications, but one has to search for the best performing algorithm on the instances of their application. The latter is an important aspect of modern data science and algorithm design. Practitioners often optimize over large families of parameterized algorithms and tune the parameters of their algorithms using a training set of problems from their domain. A famous example is the, so called, "hyperparameter optimization" of neural networks, however most of this work comes with no performance guarantees.

A recent line of research started by Roughgarden and Gupta in [GR17] and continued in many follow up papers (e.g. [BNVW17], [BDW18]), called "Data-Driven Algorithm Design", aims to put the process of selecting an application-specific algorithm on firm theoretical foundations. by using, and extending, the classic learning theory models of PAC Learning ([Val84]) and Statistical Learning Theory ([Vap99]), researchers are trying to provide strong computational and statistical performance guarantees. Focusing on specific problems, they are trying to design processes that consistently select the best-performing algorithms, among a class of alternatives, with respect to samples of domain inputs, and have optimal sample complexities.

Chawla eta al. ([CGT$^+$20]) are the first to work with non-parameterized classes of algorithms under the Data-Driven Algorithm Design setting. They revisit the Pandora's Box problem, a fundamental stochastic optimization problem, through the data-driven lens and study the correlated case and some interesting extensions. In the Pandora's Box problem, we are presented with n boxes, each containing an unknown stochastic reward. We can open boxes in any order by paying a fixed overhead for each box, observe the reward in the boxes and decide to terminate by selecting any one of the rewards observed. The goal is to maximize the reward selected minus the total overhead payed for opening boxes. Chawla et al. study a more general case where the stochastic rewards of different boxes can be arbitrarily correlated and the algorithm instead of selecting one box, has to select k boxes that form the basis of a matroid. The described setting

captures a great number of optimization problems with stochastic input. For example, graph connectivity is a special case of the above problem when one works with the Graphic Matroid. Chawla et al. define some natural classes of algorithms based on their adaptivity level and using a sufficiently large sample from the input distribution they design strategies that compete against the optimal strategies in the aforementioned classes.

Our work revisits the problem of graph connectivity under a data-driven perspective. This problem, as mentioned earlier, is captured by the matroid extension of the Pandora's Box problem when we are using the Graphic Matroid. Graph connectivity is a fundamental combinatorial optimization problem and there are many real-world cases where it appears in a stochastic probing setting. A real-world example might be a computer network, or perhaps a gas pipeline, where the connections between some nodes have been damaged or are unreliable. Given that checking the reliability of the connections may be costly, for example in the case of the gas pipeline we would have to physically visit the link's location and perform diagnostic tests, we would like to bring the network back to life by finding a spanning tree, while checking as few connections as possible.

1.1 Contributions

More formally, we are given a graph G, which we call *moldgraph*, and a distribution $\mathcal{D}$ over connected subgraphs of G. Nature chooses a graph G' from the distribution $\mathcal{D}$, whose edge set is initially hidden. We refer to G' as the *instantiated* or *realized* graph. Algorithms are allowed to probe an edge of G in order to reveal whether it exists (i.e. belongs to G') or not. Their goal is to find a spanning tree of the realized graph G', while spending as little time as possible gathering information.

Following the work of [CGT$^+$20], we define two classes of algorithms based on their adaptivity, namely the *Non-Adaptive Algorithms (NA)* and the *Adaptive Algorithms*. In the first class, algorithms decide upon a fixed set of edges that they will query regardless of the instantiated graph. The problem of finding a set that contains a spanning tree for every sampled graph is similar to the Universal TSP problem ([SS08]) and we prove that it is NP-Hard to find the minimum such set by reducing Set Cover to our problem. The reduction also preserves the inaproximability results of Set Cover, meaning that no computationally efficient Non-Adaptive algorithm can approximate the optimal Non-Adaptive cost within a factor that is less than logarithmic in the support of the distribution $\mathcal{D}$, i.e. $\Omega(\log |\mathcal{D}|)$. We also provide a randomized rounding of the Linear Programming relaxation of the Non-Adaptive algorithms, that yields an NA algorithm that is $O(\log n + \log |\mathcal{D}|)$ competitive with respect to the optimal NA cost.

The rest of our work focuses on designing Adaptive algorithms that can compete with the optimal NA cost, and perhaps break the $O(\log |D|)$ barrier. We define a noisy version of our problem which is a bottleneck case for the original problem. In this setting, The algorithms can ask a noisy oracle about whether an edge exists or not. The oracle works as follows; if the queried edge exists then it answers "Yes" with probability p and "No" with probability $1-p$. If the queried edge is not realized, ie it does not belong to $E(G')$, then it answers "No" with probability 1. In other words, if we get a positive answer we are sure that the edge exists, but when we get a negative answer we can not know whether this is a false negative or not. The goal of the

algorithm is then to minimize the expected number of queries until we reveal a spanning tree of G'. We prove that the noisy setting reduces to the original problem and its lower bounds translate to lower bounds for Adaptive algorithms in the basic problem. Furthermore, we formulate the noisy setting as a Markov Decision Process and as a result we were able to compute the optimal strategy for a given instance and thus prove a constant lower bound for Adaptive Algorithms.

Finally, we propose a specific paradigm for designing Adaptive algorithms. By taking advantage of the fact that realized graphs are connected, the strategies we design focus on specific cuts of the moldgraph and query them until they find a realized edge. These strategies can be seen as consisting of two independent parts: a subroutine that queries a given cut and a subroutine that selects the next cut to query given the answers we have received so far. In this way, translating algorithms from the noisy setting to the basic model is simple, as one only has to adjust the cut querying subroutine. We provide an optimal cut querying subroutine for both settings that are used with cut selection subroutines which are independent of the setting. Finally, we design two cut selection subroutines that achieve constant competitive ratios for specific graph families:

a) Algorithm 10 achieves an $O(k)$ competitive ratio, where k is the moldgraph's treewidth, by utilizing a tree decomposition of the moldgraph.

b) Algorithm 11 achieves an $O(\rho)$ competitive ratio for ρ-sparse and minor-closed graphs (e.g. planar graphs).

1.2 Organization

In Chapter 2, we introduce and motivate the "Data-Driven Algorithm Design" paradigm. We start by analyzing the pros and cons of worst case analysis and reason about the need of different models to analyze the performance of algorithms. We then focus on Data-Driven approaches and present the core definitions and basic tools that are widely used in this line of research. Finally, we survey some notable results including General Greedy Heuristics and Clustering Algorithms.

In Chapter 3, we discuss the correlated version of the Pandora's Box problem and its generalizations that were introduced in [CGT$^+$20]. The matroid extension of the Pandora's Box problem captures a great number of optimization problems, including graph connectivity (due to the Graphic Matroid), which is the problem we study in this thesis. In [CGT$^+$20], Chawla et. al were the first to study the correlated case and also the first to work with non-parameterized classes of algorithms under the Data-Driven Algorithm Design model. We present their core techniques and a summary of their results for all the settings they studied.

Finally, in Chapter 4 we present our work on graph connectivity under the Data-Driven Algorithm Design model. By building on [CGT$^+$20], we define some natural classes of algorithms based on their adaptivity level, namely the *Non-Adaptive* and *Adaptive* algorithms. We prove that Non-Adaptive algorithms are learnable from data and it suffices to optimize over a polynomially large sample set drawn from the input distribution. Furthermore, we prove that optimizing over Non-Adaptive algorithms is NP-hard and provide a rounding technique for their Integer Programming formulation. The rest of the work focuses on designing efficient Adaptive algorithms that are competitive against the optimal Non-Adaptive cost. We also define a noisy setting for our problem which is used to derive lower bounds for Adaptive Algorithms and is also used for

designing strategies that are later translated to algorithms for the initial problem. In the end, we list several directions for future work on our problem and some interesting directions for the field as well.

Chapter 2

Data-Driven Algorithm Design

Data-Driven Algorithm Design is a branch of a more general research field called "Beyond the Worst Case Analysis of Algorithms". Researchers in this field are trying to formulate mathematical models that explain empirically observed phenomena about algorithmic performance. However, the attempts also have an engineering dimension where the goals are to provide accurate and meaningful guidance about which algorithm to use for a problem and how to design algorithms that work well on particular inputs.

In this section we will discuss the motivations behind *Data-Driven Algorithm Design* and present some of the field's core concepts. The analysis in this chapter follows [Rou21].

2.1 Beyond the Worst Case Analysis of Algorithms

Comparing different algorithms is hard. For almost any pair of algorithms and any measure of algorithmic performance, each algorithm will perform better than the other on some inputs. For example, `MergeSort` will need $O(n \log n)$ steps in order to sort an array of length n, regardless of whether that array is already sorted or not. On the contrary, `InsertionSort` will need only $O(n)$ steps if the input array is already sorted. In general, when two algorithms have incomparable performance, or have their own regions of dominance, how can we deem one better than the other?

2.1.1 The Benefits of Worst-Case Analysis

Worst-case analysis is a specific modeling choice in the analysis of algorithms, where the performance profile of an algorithm is summarized by its worst performance on any input of a given size. The better algorithm is the one with superior worst-case performance. `MergeSort`, with its worst-case performance of $O(n \log n)$, is in this sense better than `InsertionSort`, that has a worst-case performance of $O(n^2)$.

While crude, worst-case analysis can be tremendously useful and, for several reasons, it has been the dominant paradigm for algorithm analysis in theoretical computer science.

1. A good worst-case guarantee is strongest result one can prove for an algorithm. It certifies its general-purpose utility and relieves its users from understanding which inputs are most

relevant to their applications. Therefore, worst-case analysis is particularly well suited for "general-purpose" algorithms that are expected to work well across a range of application domains.

2. The worst-case performance of an algorithm is often easier to be calculated than its alternatives, such as average-case analysis with respect to a probability distribution over inputs.

3. For a large number of fundamental combinatorial optimization problems, there already exist algorithms that have excellent worst-case guarantees, deeming the search for alternatives unnecessary.

2.1.2 Goals of the Analysis of Algorithms

Before critiquing worst-case analysis, we should take a step back and ponder: What exactly is our goal when we are trying to reason about algorithm performance? T. Roughgarden ([[Rou21]]) identifies three such goals:

Performance prediction. The first goal is to be able to explain or predict the empirical performance of algorithms. In some cases, taking an observed phenomenon as ground truth (e.g. that the LRU policy performs reasonably well), we are seeking a transparent mathematical model that explains it. In other cases, we might be seeking for a theory that will give accurate advice about whether or not an algorithm will perform well in an application of interest.

Identifying optimal algorithms. The second goal is to rank different algorithms according to their performance, and ideally to single out one algorithm as "optimal". At the very last, given two algorithms A and B for the same problem, a method of algorithmic analysis should offer an opinion about which one is "better".

Developing new algorithms. Finally, we would like a well-defined framework in which to brainstorm new algorithms.

2.1.3 The Cons of Worst-Case Analysis

We should celebrate the fact that worst-case analysis has worked so well for so many fundamental combinatorial problems, but at the same time we should recognize that its successes may paint a misleading picture about the range of its practical revelance.

Overly Pessimistic Performance Predictions. As evident by its name, worst case analysis gives a pessimistic estimate of an algorithm's empirical performance. A very famous example is the Simplex Method for Linear Programming that was proposed by Dantzig in 1947. Its running time typically scales modestly with the input size but its exponential worst case complexity proved by Klee and Minty in [KM70] comes at odds with its robust empirical performance.

Inaccurate Ranking of Algorithms. Overly pessimistic summaries can derail worst case analysis from identifying the right algorithm to use in practice. For instance, it can not distinguish between the FIFO and LRU policies in online paging and it also suggests that the ellipsoid method is better than the empirically superior simplex method for linear programming.

No data model. Worst-case analysis has an implicit model of data, where the instance to be solved is an adversarially selected function of the chosen algorithm. Outside of security applications, this algorithm-dependent model of data is a rather paranoid and incoherent way to think about a computational problem. After all, the properties that the input data exhibit are what distinguishes good performing algorithms from poor ones. For instance, in the aforementioned case of online paging, the LRU policy is empirically better because in practical applications (e.g. memory requests from computer programs), requests tend to exhibit the so-called "locality of reference", meaning that recently requested pages are more likely to be requested again.

These drawbacks show the importance of alternatives to worst-case analysis, in the form of models that articulate properties of "relevant" inputs and algorithms that possess rigorous and meaningful algorithmic guarantees for inputs with these properties.

2.2 Data-Driven Approaches

In an effort to design algorithms for real-world problems, instead of using off the self algorithms with weak worst case guarantees, practitioners often employ a data-driven approach: they use past instances of their problem and try to learn a method that works well in their particular domain. This idea has long been used in practice in various communities, including theory ([Fin98, ACCL06]), artificial intelligence ([HRG+13, XHHLB08]), computational biology ([DK18]), and auction design ([San03]). However, so far, most of this work has come with no performance guarantees. By building on learning theory tools, recent work on this area provides the first formal guarantees for data-driven approaches. A great introductory survey of these attempts has been written by Maria-Florina Balcan in [Bal21].

2.2.1 Algorithm design as a distributional learning problem

Tim Roughgarden and Rishi Gupta ([GR15]) proposed to analyze the data-driven algorithm design problem as a distributional learning problem, by using, and extending, the classic learning theory models of PAC Learning ([Val84]) and Statistical Learning Theory ([Vap99]). In this framework, the application domain is modeled as an unknown distribution over problem instances to which algorithms have sample access during an initial learning phase. The formal guarantees we aim for are generalization guarantees that quantify how many training problem instances are needed to ensure that an algorithm with good performance over the training instances will exhibit good performance on future problem instances.

Problem Formulation by [Bal21]. We fix an algorithmic problem (e.g. a subset selection problem or a clustering problem) and we denote by Π the set of problem instances of interest for

this problem. We also fix a large (potentially infinite) family of algorithms $\mathcal{A}$, and we assume that this family is parameterized by a set $\mathcal{P} \subseteq \mathbb{R}^d$; we denote by A_ρ the algorithm in $\mathcal{A}$ parameterized by ρ. We also fix a utility function $u : \Pi \times \mathcal{P} \to [0, H]$, where $u(x, \rho)$ measures the performance of the algorithm A_ρ on problem instance $x \in \Pi$. We denote by $u_\rho(\cdot)$ the utility function $u_\rho : \Pi \to [0, H]$ induced by A_ρ, where $u_\rho(x) = u(x, \rho)$. Note that u is bounded; for example, for cases where u is related to an algorithm's running time, H can be the time-out deadline.

The "application-specific information" is modeled by the unknown distribution $\mathcal{D}$. The learning algorithm is given i.i.d. samples $x_1, \ldots, x_m \in \Pi$ from $\mathcal{D}$, and (perhaps implicitly) the corresponding performance $u_\rho(x)$ of each algorithm $A_\rho \in \mathcal{A}$ on each input x_i. The learning algorithm uses this information to suggest an algorithm $A_\rho \in \mathcal{A}$ to use on future inputs drawn from $\mathcal{D}$. We seek learning algorithms that almost always output an algorithm of $\mathcal{A}$ that performs almost as well as the optimal algorithm A_{ρ^*} for $\mathcal{D}$ that maximizes $\mathbf{E}_{x \sim \mathcal{D}}[u_\rho(x)]$ over $A_\rho \in \mathcal{A}$.

2.2.2 The Pseudo-dimension of Algorithms

To achieve the desired guarantees, we rely on uniform convergence results. To be more specific, we want to quantify how many training instances are needed in order to guarantee that with high probability the empirical performance of all algorithms in $\mathcal{A}$ will be close to their expected performance on the distribution $\mathcal{D}$. It is known from learning theory, that these results depend on the intrinsic complexity of the family of real-valued utility functions $\{u_\rho(\cdot)\}_\rho$. The notion that is used as a rough measure of the complexity of those families, is the so-called *pseudo-dimension*, that quantifies the ability of the class to fit complex patterns.

Definition 2.2.1 (Definition 2.1 of [Bal21], Pseudo-dimension)**.** Let $\{u_\rho(\cdot)\}_\rho$ be the family of performance measures induced by $\mathcal{A} = \{A_\rho\}_\rho$ and the utility function $u(x, \rho)$.

(a) Let $\mathcal{S} = \{x_1, \ldots, x_m\} \subset \Pi$ be a set of problem instances and let $z_1, \ldots, z_m \in \mathbb{R}$ be a set of targets. We say that $z_1, \ldots, z_m$ witness the shattering of $\mathcal{S}$ by $\{u_\rho(\cdot)\}_\rho$ if for all subsets $T \subseteq \mathcal{S}$, there exists some parameter $\rho \in \mathcal{P}$ such that for all elements $x_i \in T,\ u_\rho(x_i) \leq z_i$ and for all $x_i \notin T,\ u_\rho(x_i) > z_i$. We say that $\mathcal{S}$ is shattered by $\{u_\rho(\cdot)\}_\rho$ if there exist $z_1, \ldots, z_m$ that witness its shattering.

(b) Let $\mathcal{S} \subseteq \Pi$ be the largest set that can be shattered by $\{u_\rho(\cdot)\}_\rho$. The *pseudo-dimension* of the class $\{u_\rho(\cdot)\}_\rho$ is $|\mathcal{S}|$.

When $\{u_\rho(\cdot)\}_\rho$ is a set of binary valued functions, the notion of pseudo-dimension reduces to the notion of VC-dimension.

Similarly to Learning Theory and VC-dimension, to obtain sample complexity guarantees in data-driven algorithm selection, it suffices to bound the pseudo-dimension of the family $\{u_\rho(\cdot)\}_\rho$. The following theorem formulates the former statement and can be seen as an extension of the Fundamental Theorem of Statistical Learning Theory.

Theorem 2.2.2 (Theorem 2.2 of [Bal21])**.** *Let $d_{\mathcal{A}}$ be the pseudo-dimension of the family of utility functions $\{u_\rho(\cdot)\}_\rho$ induced by the class of algorithms $\mathcal{A}$ and the utility function $u(x, \rho)$; assume*

that the range of $u(x, \rho)$ is $[0, H]$. For any $\epsilon > 0$, any $\delta \in (0, 1)$ and any distribution $\mathcal{D}$ over Π, $m = O\left(\frac{H^2}{\epsilon^2}(d_{\mathcal{A}} + \ln \frac{1}{\delta})\right)$ samples are sufficient to ensure that with probability $1 - \delta$ over the draw of m samples $\mathcal{S} = \{x_1, \ldots, x_m\} \sim D^m$, for all $\rho \in \mathcal{P}$, the difference between the average utility of the algorithm A_ρ over the samples and its expected utility is less than ϵ, that is

$$\left| \frac{1}{m} \sum_{i=1}^{m} u_\rho(x_i) - \boldsymbol{E}_{x \sim \mathcal{D}} \left[u_\rho(x) \right] \right| \leq \epsilon$$

Theorem 2.2.2 implies that to obtain sample complexity guarantees it is sufficient to bound the pseudo-dimension of $\{u_\rho(x)\}_{\rho \in \mathcal{P}}$. Interestingly, many of the proofs in the literature achieve this by providing a structural result for the dual class of functions $\{u_x(\rho)\}_{x \in \Pi}$, where $u_x(\rho) = u(x, \rho) = u_\rho(x)$. For example, the following lemma is a fundamental result that has been used in several papers.

Lemma 2.2.3 (Lemma 29.3 of [Bal21])**.** *Suppose that for every instance $x \in \Pi$, the function $u_x(\rho) : \mathbb{R} \to \mathbb{R}$ is piece-wise constant with at most N pieces. Then the family $\{u_\rho(x)\}_{\rho \in \mathcal{P}}$ has pseudo-dimension $O(\log N)$.*

Proof of Lemma 2.2.3. Consider a problem instance $x \in \Pi$. Since the function $u_x(\rho)$ is piece-wise constant with at most N pieces, there are at most $N - 1$ critical points $\rho_1^*, \rho_2^*, \ldots$, such that between any two consecutive critical points ρ_i^*, ρ_{i+1}^* the function $u_x(\rho)$ is constant.

Consider m problem instances $x_1, \ldots, x_m$. Taking the union of their critical points and sorting them, between any two consecutive of these critical points we have that *all* of the functions $u_{x_j}(\rho)$ are constant. Since these critical points break up the real line into at most $(N-1)m+1 \leq Nm$ intervals, and all $u_{x_j}(\rho)$ are constant in each interval, this means that overall there are at most Nm different m-tuples of values produced over all ρ. Equivalently, the functions $u_\rho(x)$ produce at most Nm different m-tuples of values on the inputs $x_1, \ldots, x_m$. However, to shatter the m instances, we must have 2^m different m-tuples of values produced. Solving $Nm \geq 2^m$ shows that only sets of instances of size $m = O(\log N)$ can be shattered. □

2.2.3 The pseudo-dimension of greedy heuristics

In this section we present some results regarding infinite parameterized families of greedy algorithms for subset selection problems, as they were introduced and analyzed in [GR15, GR17].

Knapsack. As an introductory example, we will talk about the knapsack problem. Let $\mathcal{A}_{knapsack} = \{\mathcal{A}_\rho\}$ be the family of greedy algorithms. For this family, $\mathcal{P} = \mathbb{R}_{\geq 0}$, and for $\rho \in \mathcal{P}$, for an instance x where v_i and s_i are the value and size of item i, the algorithm A_ρ adds the items to the knapsack in decreasing order of v_i / s_i^ρ subject to the capacity constraint. The utility function $u(x, \rho)$ is defined as the value of the items chosen by the greedy algorithm with parameter ρ on the input x. We can show that the class $\mathcal{A}_{knapsack}$ is not too complex, in the sense that its pseudo-dimension is small.

Theorem 2.2.4 (Theorem 29.4 of [Bal21])**.** *The family of utility functions $\{u_\rho(x)\}$ corresponding $\mathcal{A}_{knapsack}$ has pseudo-dimension $O(\log n)$, where n is the maximum number of items in an instance.*

Proof of Theorem 2.2.4. We first show that each function $u_x(\rho)$ is piecewise constant with at most n^2 pieces and then apply Lemma 2.2.3.

To show the first part, fix some instance x. Now, suppose algorithm A_{ρ_1} produces different solution on x than A_{ρ_2} does for $\rho_1 < \rho_2$. We argue there must exist some critical value $c \in [\rho_1, \rho_2]$ and some pair of items $i, j \in x$ such that $v_i/s_i^c = v_j/s_j^c$. The reason is that if A_{ρ_1} and A_{ρ_2} produce different solutions on x, they must at some point make different decisions about which item to add to the knapsack. Consider the first point where they differ: say that A_{ρ_1} adds item i to the knapsack and A_{ρ_2} adds item j. Then it must be the case that $v_i/s_i^{\rho_1} - v_j/s_j^{\rho_2} \geq 0$ but $v_i/s_i^{\rho_2} - v_j/s_j^{\rho_2} \leq 0$. Since the function $f(\rho) = v_i/s_i^{\rho} - v_j/s_j^{\rho}$ is continuous, there must exist some value $c \in [\rho_1, \rho_2]$ such that $v_i/s_i^c - v_j/s_j^c$ as desired.

Now, for any given pair of items i, j there is at most one value of $\rho \geq 0$ such that $v_i/s_i^{\rho} = v_j/s_j^{\rho}$; indeed, it is $\rho = \log(v_i/v_j)/log(s_i/s_j)$. This means there are at most $\binom{n}{2}$ critical values c such that $v_i/s_i^c - v_j/s_j^c$ for some pair of items $i, j \in x$. By the preceding arguemnt, all values of ρ in the interval between any two consecutive critical values must produce the same behavior on the instance x. This means that there are at most $\binom{n}{2} + 1 \leq n^2$ intervals such that all values of ρ inside the same interval result in the exact same solution by A_{ρ}.

Finally, we can apply Lemma 2.2.3 with $N = n^2$ which concludes the proof. □

A General Analysis for Greedy Heuristics. We know more generally consider problems in which the input is a set of n objects with various attributes, and the feasible solutions consist of assignments of the objects to a finite set , subject to feasibility constraints. The attributes of an object are represented as an element ξ of an abstract set. For example, in the Knapsack problem, ξ encodes the value and size of an object and $Y = \{0, 1\}$, indicating whether or not a given object is selected.

Roughgarden et al ([GR17]) provide pseudo-dimension bounds for general greedy heuristics of the following form:

1. Use a scoring rule σ (a function from attributes to $\mathbb{R}$) to compute a score $\sigma(\xi_i)$ for each object i, as a function of its current attributes ξ_i.

2. For the unassigned object i with the highest score, use an assignment rule to assign i a value from Y and, if necessary, update the attributes of the other unassigned objects. Assume that ties are always resolved lexicographically.

In a *single-parameter* family of scoring rules, there is a scoring rule of the form $\sigma(\rho, \xi)$ for each parameter value ρ in some interval $I \subseteq \mathbb{R}$. Moreover, σ is assumed to be continuous in ρ for every fixed value of ξ.

A single parameter family of scoring rules is κ-crossing if, for each distinct pair of attributes ξ', ξ'', there are at most κ values of ρ for which $\sigma(\rho, \xi') = \sigma(\rho, \xi'')$. For example, the scoring rule mentioned for Knapsack is 1-crossing, as for two items i and j there exists a unique ρ_0 such that $v_i/s_i^{\rho_0} = v_j/s_j^{\rho_0}$. Moreover, if the distinct number of values that an assignment rule assigns items is at most β, then it is called β-bounded. Combining a β-bounded assignment rule and a κ-crossing scoring rule yields a (κ, β)-single-parameter family of greedy heuristics.

Theorem 2.2.5 (Theorem 29.5 of [Bal21]). *Let $\mathcal{A}_{greedy}$ be a (κ, β) single parameter family of greedy heuristics and let $\{u_\rho(x)\}$ be its corresponding family of utility functions. The pseudo-dimension of $\{u_\rho(x)\}$ is $O(\log(\kappa\beta n))$, where n is the number of objects.*

Interestingly, the former theorem indicates that all logical greedy heuristics that follow continuous scoring rules with limited crossings are not too complex classes of algorithms, in the sense that they have low pseudo-dimension.

2.2.4 Clustering Problems

In this section we discuss how a data-driven approach can help overcome impossibility results for clustering problems. Clustering is one of the most basic problems in data science, where given a large set of complex data (e.g. images or news articles) the goal is to organize it into groups of similar items. Almost all natural optimization problems that are defined over clusterings are NP-hard. However, in practice, clustering is not viewed as a particularly difficult problem. Lightweight clustering algorithms, such as Lloyd's algorithm for k-means and its variants, regularly return the intuitively "correct" clustering of real-world point sets. How can we reconcile the worst-case intractability of clustering problems with the empirical success of relatively simple algorithms?

There have been several attempts to address the former question. One approach, which is surveyed in [MM21, Blu21], is to posit specific stability assumptions about the input instances and to design efficient algorithms with good performance on such instances. Another approach that is particularly suited for settings (including text and image categorization) where we have to solve many clustering problems arising in a given application domain, is to select a good clustering algorithm in a data-driven way. In particular, given a series of clustering instances to be solved from the same domain, we learn a good parameter setting for a clustering algorithm that performs well on instances coming from that domain.

Problem Setup The input to a clustering problem is a point set V of n points, a desired number of clusters $k \in \{1, \ldots, n\}$, and a metric d (such as the Euclidean distance in $\mathbb{R}^d$) specifying the distance between any two points. Throughought the rest of this section we will denote by $d(i, j)$ the distance between points i and j. The goal is typically to output a partition $\mathcal{C} = \{C_1, \ldots, C_k\}$ of V that optimizes a specific objective function. For example, in the k-means clustering we output a center for c_i for each $C_i \in \mathcal{C}$ and the objective is to minimize the sum of the squared distances between every point and its nearest center, i.e. $cost(\mathcal{C}) = \sum_i \sum_{v \in C_i} d(v, c_i)^2$, while in the k-median objective the goal is to minimize the sum of distances to the centers, i.e. $cost(\mathcal{C}) = \sum_i \sum_{v \in C_i} d(v, c_i)$

Linkage-Based Families We will now discuss some families of two-stage clustering algorithms, that in the first stage use a linkage procedure to organize data into a hierarchical clustering and then in the second stage use a fixed (and computationally efficient) procedure to extract a pruning from the hierarchy.

The linkage procedure in the first step takes as input a clustering instance x and outputs a cluster tree, by repeatedly merging the two closest clusters. In particular, starting with the base

distance d, we first define a distance measure $D(A, B)$ between any two subsets A and B of $\{1, \ldots, n\}$, that is used to greedily link the data into a binary cluster tree. The leaves of the tree are individual data points, while the root node corresponds to the entire dataset. The algorithm starts with each point belonging to its own cluster. Then, it repeatedly merges the closest pair of clusters according to distance D. When there is only a single cluster remaining, the algorithm outputs the constructed cluster tree. Different definitions for D lead to different hierarchical procedures. Some widely-used variants are the following.

Single Linkage	$D(A, B) = \min_{a \in A, b \in B} d(a, b)$
Complete Linkage	$D(A, B) = \max_{a \in A, b \in B} d(a, b)$
Average Linkage	$D(A, B) = \frac{1}{\lvert A \rvert \lvert B \rvert} \sum_{a \in A, b \in B} d(a, b)$

The procedure in the second step can be as simple as just "undoing" the last $k - 1$ merges from the first step or as complex as a dynamic programming subroutine over the hierarchy to extract a clustering of highest score based on some measureable objective such as k-means or k-median cost. The final quality of the solution (measured by $u_\rho(x)$ on clustering instance x) produced by the algorithm is measured by the given objective function.

Balcan et al. ([BNVW17]) analyze the pseudo-dimension of two parametric families of algorithms of this form. Both of these families use a parametric linkage procedure in the first step, and the cluster tree produced is then fed into a fixed second-stage procedure to produce a k-clustering. The first family, $\mathcal{A}^{scl}$ uses a parameterized family of algorithms with a single parameter $\rho \in \mathcal{P} = [0, 1]$ that helps interpolate linearly between the classic single and complete linkage. For $\rho \in \mathcal{P}$, the algorithm $A_\rho \in \mathcal{A}^{scl}$ defines the distance between two sets A and B as

$$D_\rho^{scl}(A, B) = (1 - \rho)\, d_{\min}(A, B) + \rho\, d_{\max}(A, B)$$

The second family, $\mathcal{A}^{exp}$, uses a parameterized family of linkage algorithms with a single parameter $\rho \in \mathcal{P} = \mathbb{R}$ that helps interpolate not only between single and complete linkage but also includes average linkage as well. For $\rho \in \mathcal{P}$ the algorithm $A_\rho \in \mathcal{A}^{exp}$ defines the distance between two sets A and B as

$$D_\rho^{exp}(A, B) = \left(\frac{1}{\lvert A \rvert \lvert B \rvert} \sum_{u \in A, v \in B} (d(u, v))^\rho \right)^{1/\rho}$$

Note that $\rho = 0$ recovers average linkage, $\rho \to \infty$ recovers complete linkage and $\rho \to -\infty$ recovers single linkage.

Balcan et al. ([BNVW17]) prove the following results.

Lemma 2.2.6 (Lemma 29.6 of [Bal21]). *Let x be a clustering instance. We can partition $\mathcal{P}$ into at most n^8 intervals such that all values of ρ inside the same interval result in the exact same solution produced by the D_ρ^{scl}-linkage algorithm.*

Lemma 2.2.6 combined with Lemma 2.2.3 imply the following two results

Theorem 2.2.7 (Theorem 29.7 of [Bal21]). *The family of functions $\{u_\rho(x)\}$ corresponding to the family of $\mathcal{A}^{scl}$-linkage has pseudo-dimension $O(\log n)$.*

Theorem 2.2.8 (Theorem 29.8 of [Bal21]). *The family of functions $\{u_\rho(x)\}$ corresponding to the family of $\mathcal{A}^{exp}$-linkage has pseudo-dimension $O(n)$.*

Interestingly, these families of clustering algorithms are also known to have strong analytical properties for stable instances ([BDR14, BL16, BW17, BHW20]). One such condition, called pertrubation-resilience, asks that even if distances between data points are pertrubed by up to some factor β, the clustering that optimizes a given objective (such as k-means or k-median) does not change. If this condition is satisfied for $\beta \geq 2$, it is known that one can find the optimal clustering efficiently, in fact via a linkage algorithm followed by dynamic programming, further motivating that algorithm family. However, one drawback of all these results is that if the condition does not hold, the guarantees do not apply. Here, we aim to provide guarantees on optimality within an algorithm family that hold regardless of clusterability assumptions, but with the additional property that if typical instances are indeed well-clusterable (e.g. they satisfy pertrubation-resilience or some related condition), then the optimal algorithm in the family is optimal overall. This way, we can produce guarantees that simultaneously are meaningful in the general case and can take advantage of settings in which the data are particularly well behaved.

2.2.5 Other Applications

Learning to branch. So far we considered families of polynomial-time algorithms and scored them based on solution quality (e.g. clustering quality or objective value). In general, one could also score algorithms based on other important measures of performance. For example, [BDSV18] consider parameterized branch-and-bound techniques for learning how to branch when solving mixed integer programs (MIPs) in the distributional learning setting, and score a parameter setting based on the tree size on a given instance (which roughly corresponds to running time). Balcan et al. show that the corresponding dual functions are piecewise constant, and then the sample complexity results follow from a high-dimensional generalization of Lemma 2.2.3. Balcan et al. also show experimentally that different parameter settings of these families of algorithms can result in branch and bound trees of vastly different sizes, for different combinatorial problems (including winner determination in combinatorial auctions, k-means clustering, and agnostic learning of linear separators). They also show that the optimal parameter is highly distribution dependent: using a parameter optimized on the wrong distribution can lead to a dramatic tree size blowup, implying that learning to branch is both practical and hugely beneficial.
General Theorem. Balcan et al ([BDD$^+$21]) present a general sample complexity result applicable to algorithm configuration problems for which the dual functions are piece-wise structured. The key innovation is to provide an elegant and widely applicable abstraction that simultaneously covers all the types of dual structures appearing in the algorithm families mentioned so far.

They show that this theorem recovers all the prior results and they also show new applications including dynamic programming techniques for important problems in computational biology, e.g., sequence alignment and protein folding.Finally, another notable case that has been studied is automated Mechanism Design ([MR15, BSV16, BSV18]).

2.2.6 Open Directions

Data-Driven algorithm design has the potential to fundamentally shift the way we analyze and design algorithms for combinatorial problems. In addition to scaling up the techniques developed so far and also using them for new problems, it would be interesting to develop new analysis frameworks that lead to even better automated algorithm design techniques. For example, it would be interesting to explore a reinforcement learning approach, where we would learn state-based decision policies that use properties of the current state of the algorithm to determine how to proceed. It would also be interesting to develop tools for learning within a single problem instance as opposed to learning across instances.

Chapter 3

Pandora's Box Problem

In this section we provide a short summary of [CGT+20] which is the main paper our work builds on. The latter is the first work in Data-Driven Algorithm Design were the classes of Algorithms examined are not parameterized. Chawla et al. define some general and natural non-parameterized classes of Algorithms which they use as benchmarks for the algorithms they design. The "application-specific" part is still modeled by an unknown distribution $\mathcal{D}$ over problem instances, but instead of calculating sample complexities that will guarantee that an empirically good performing algorithm will exhibit good average performance on the distribution $\mathcal{D}$, the focus is on using samples of $\mathcal{D}$ to design good proper and non-proper algorithms. In an effort to provide some guarantees about the adaptivity of these algorithms to the distribution $\mathcal{D}$, the authors prove that they are competitive to the best performing algorithms of some classes of algorithms with respect to the distribution $\mathcal{D}$.

3.1 Problem Definition

The Pandora's Box problem was first introduced by Weitzman in [Wei79], and has a long literature of generalizations ([CFG+02, GGM06, ASW13, GNS, GJSS19]). It captures a great variety of optimization problems with stochastic input where the algorithm can obtain information of input random variables at some cost. Determining the optimal manner for acquiring information becomes an online decision-making problem, since each piece of information revealed to the algorithm can affect its decisions. The Pandora's Box is essentially a search problem where, while trying to find the cheapest among a set of alternatives, we should not spend too much time searching for it. A real-world example might be the process of searching for a home to rent. We have many available houses as options, but in order to find out how much we like a house and how it compares to other houses we have seen, we have to arrange a meeting and physically visit the location. Ideally, we would like to select the house that best fits our desires while not spending too much time visiting different sites.

More formally, in the Pandora's Box problem, the algorithm is presented with n boxes, each containing an unknown stochastic reward. The algorithm can open boxes in any order by paying a cost for each and can decide to terminate at any time upon selecting any of the observed rewards. The goal is to maximize the reward selected minus the total overhead of opening boxes.

A crucial assumption made by all works prior to [CGT+20] is that the random variables corresponding to rewards in different boxes are independent. However, this does not always

bear out in practice. For instance, in the home renting example, houses in the same neighborhood might have similar characteristics and our valuations for them might be correlated. Chawla et al., are the first to study the Pandora's Box problem with correlations and assume that the instantiated results (henceforth scenarios) are drawn from a joint distribution $\mathcal{D}$.

3.2 Natural classes of Algorithms based on Adaptivity

The optimal solution for the online stochastic search would be a fully-adaptive strategy that chooses which box to query each time based on all the costs that have been observed so far. While these are the best strategies one could hope for, they are impossible to find or approximate with samples. For example it could be the case that the cost in the first few boxes encode the location of a box of cost 0 while every other box has infinite cost. While the best option can be identified with just a few queries, any reasonable approximation of the optimal cost, would need to accurately learn this mapping. Learning such an arbitrary mapping however is impossible through samples, unless there is significant probability of seeing the exact same combination of costs. Therefore, the authors define some general, non-parameterized classes of Algorithms that they use as benchmarks. Any online algorithm can be described by the pair (σ, τ) where σ is a permutation of the boxes representing the order in which they get probed, and τ is a stopping rule – the time at which the algorithm stops probing and returns the minimum cost it has seen so far. All algorithms are divided in the following classes.

Definition 3.2.1 (Definition 2.1 of [CGT$^+$20], Adaptivity of Strategies)**.** In a *Fully-Adaptive (FA) strategy*, both σ and τ can depend on any costs seen in a previous time step, as described above.

In a *Partially-Adaptive (PA) strategy*, the sequence σ is independent of the costs observed in probed boxes. The sequence is determined before any boxes are probed. However, the stopping rule τ can depend on the identities and costs of boxes probed previously.

In a *Non-Adaptive (NA) strategy*, both σ and τ are fixed before any costs are revealed to the algorithm. In particular, the algorithm probes a fixed subset of the boxes and returns the minimum cost observed.

3.3 A reduction to scenario-aware strategies and its implications to learning

Designing a PA strategy involves determining a non-adaptive probing order and a good stopping rule for that probing order. Since the problem does not place any bounds on the number of different scenarios, m, or the support size and range of the boxes' costs, these numbers can be exponential or even unbounded. Therefore, the optimal stopping rule can be very complicated and it appears to be challenging to characterize the set of all possible PA strategies. Chawla et al. simplify the optimization problem by providing *extra power* to the algorithm and then removing this power at a small loss in approximation factor.

In particular, they define a *Scenario-Aware Partially-Adaptive (SPA)* strategy as one where the probing order σ is independent of the costs observed in probed boxes, however, the stopping

time τ is a function of the instantiated scenario s. In other words, the algorithm fixes a probing order, then learns of the scenario instantiated, and then determines a stopping rule for the chosen probing order based on the revealed scenario.

Observe that once a probing order and instantiated scenario are fixed, it is trivial to determine an optimal stopping time in a scenario aware manner. The problem therefore boils down to determining a good probing order. The space of all possible SPA strategies is also likewise much smaller and simpler than the space of all possible PA strategies. We can therefore argue that in order to learn a good SPA strategy, it suffices to optimize over a small sample of scenarios drawn randomly from the underlying distribution. We denote the cost of an SPA strategy with probing order σ by $cost(\sigma)$.

On the other hand, we argue that scenario-awareness does not buy much power for the algorithm. In particular, given any fixed probing order, we can construct a stopping time that depends only on the observed costs, but that achieves a constant factor approximation to the optimal scenario-aware stopping time for that probing order.

3.3.1 Ski Rental with varying buy costs

The authors exhibit a connection between their problem and a generalized version of the ski rental problem to show that PA strategies are competitive against SPA strategies.

The input to the generalized version of the ski rental problem is a sequence of non-increasing buy costs, $a_1 \geq a_2 \geq a_3 \geq \ldots$. These costs are presented one at a time to the algorithm. At each step t, the algorithm decides to either rent skis at a cost of 1, or buy skis at a cost of a_t. If the algorithm decides to buy, then it incurs no further costs for the remainder of the process. Observe that an offline algorithm that knows the entire cost sequences $a_1, a_2, \ldots$ can pay $\min_{t\geq 1}(t-1+a_t)$. We call this problem *ski rental with time-varying buy costs*. The original ski rental problem is the special case where $a_t = B$ or 0 from the time we stop skiing and on.

The authors provide a randomized algorithm that achieves a competitive ratio of $e/(e-1)$. Their algorithm uses the randomized algorithm of [KMMO90] for ski-rental as a building block, by essentially starting a new instance of ski rental every time the cost of the offline optimum changes. With this result they arrive at the following Corollary.

Corollary 3.3.1 (Corollary 3.2 of [CGT$^+$20])**.** Given any *scenario-aware partially-adaptive* strategy σ, we can efficiently construct a stopping time τ, such that the cost of the *partially-adaptive* strategy (σ, τ) is no more than a factor of $e/(e-1)$ times the cost of σ.

Corollary 3.3.1 connects scenario-aware partially-adaptive strategies with partially-adaptive strategies through the algorithm for the ski-rental with varying costs problem. Specifically, given an SPA strategy, we can construct an instance of the ski-rental problem, where the buy cost a_t at any step is equal to the cost of the best feasible solution seen so far by the SPA strategy. The rent cost of the ski rental instance reflects the probing time of the search algorithm, whereas the buy cost reflects the cost of the boxes chosen by the algorithm. Our algorithm for ski rental chooses a stopping time as a function of the costs observed in the past and without knowing the (scenario-dependent) costs to be revealed in the future and therefore gives us a PA strategy for the search problem.

3.3.2 Learning a good probing order

After Corollary 3.3.1, the focus is now shifted on designing good scenario-aware partially adaptive strategies for the search problem. As noted previously, once we fix a probing order, determining the optimal scenario-aware stopping time is easy. The authors go on to show that in order to optimize over all possible probing orders, it suffices to optimize with respect to a small set of scenarios drawn randomly from the underlying distribution.

Formally, let $\mathcal{D}$ denote the distribution over scenarios and let $\mathcal{S}$ be a collection of m scenarios drawn independently from $\mathcal{D}$, with m being a large enough polynomial in n. Then, we claim that with high probability, for *every* probing order σ, $cost_{\mathcal{D}}(\sigma)$ is close to $cost_{\mathcal{S}}(\sigma)$, where $cost_{\mathcal{D}}(\sigma)$ denotes the total expected cost of the SPA strategy σ over the scenario distribution $\mathcal{D}$, and $cost_{\mathcal{S}}(\sigma)$ denotes its cost over the uniform distribution over the sample $\mathcal{S}$. The implication is that it suffices for us to optimize for SPA strategies over scenario distributions with finite small support.

Lemma 3.3.2 (Lemma 3.3 of [CGT$^+$20])**.** *Let $\epsilon, \delta > 0$ be given parameters. Let $\mathcal{S}$ be a set of m scenarios chosen independently at random from $\mathcal{D}$ with $m = poly(n, 1/\epsilon, \log(1/\delta))$. Then, with probability at least $1 - \delta$, for all permutations $\pi : [n] \to [n]$, we have*

$$cost_{\mathcal{S}}(\pi) \in (1 \pm \epsilon) cost_{\mathcal{D}}(\pi).$$

Combining Corollary 3.3.1 and Lemma 3.3.2 yields the following theorem.

Theorem 3.3.3 (Theorem 3.4 of [CGT$^+$20])**.** *Suppose there exists an algorithm for the optimal search problem that runs in time polynomial in the number of boxes n and the number of scenarios m, and returns an SPA strategy achieving an α-approximation. Then, for any $\epsilon > 0$, there exists an algorithm that runs in time polynomial in n and $1/\epsilon$ and returns a PA strategy with competitive ratio $\frac{e}{e-1}(1+\epsilon)\alpha$, where $n = |\mathcal{B}|$.*

3.3.3 LP Formulations of SPA and NA strategies

We will now present the LP relaxation that the authors construct for the optimal scenario-aware partially adaptive strategy. Following Lemma 3.3.2, they focus on the setting where the scenario distribution is uniform over a small support set $\mathcal{S}$.

The Linear Program given below is similar to the one used for the Generalized Min Sum Set Cover problem in [BGK10] and [SW11].

Denote by $\mathcal{T}$ to set of time steps. Let x_{it} be an indicator variable for whether box i is opened at time t. Constraints (3.1) and (3.2) model matching constraints between boxes and time slots. The variable z_{ist} indicates whether box i is selected for scenario s at time t. Constraints (3.3) ensure that we only select opened boxes. Constraints (3.4) ensure that for every scenario we have selected exactly one box. The cost of the box assigned to scenario s is given by $\sum_{i,t} z_{ist} c_{is}$. Furthermore, for any scenario s and time t, the sum $\sum_i z_{ist}$ indicates whether the scenario is covered at time t, and therefore, the probing time for the scenario is given by $\sum_t \sum_i t z_{ist}$.

$$\text{minimize} \quad \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{B}, s \in \mathcal{S}, t \in \mathcal{T}} t z_{ist} + \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{B}, s \in \mathcal{S}, t \in \mathcal{T}} c_{is} z_{ist} \tag{LP-SPA}$$

$$
\begin{aligned}
\text{subject to} \quad & \sum_{i \in \mathcal{B}} x_{it} = 1, && \forall t \in \mathcal{T} && (3.1)\\
& \sum_{t \in \mathcal{T}} x_{it} \leq 1, && \forall i \in \mathcal{B} && (3.2)\\
& z_{ist} \leq x_{it}, && \forall s \in \mathcal{S}, i \in \mathcal{B}, t \in \mathcal{T} && (3.3)\\
& \sum_{t' \in \mathcal{T}, i \in \mathcal{B}} z_{ist'} = 1, && \forall s \in \mathcal{S} && (3.4)\\
& x_{it}, z_{ist} \in [0,1] && \forall s \in \mathcal{S}, i \in \mathcal{B}, t \in \mathcal{T}
\end{aligned}
$$

The corresponding relaxation (LP-NA) for the optimal NA strategy is simpler. Here x_i is an indicator variable for whether box i is opened and z_{is} indicates whether box i is assigned to scenario s.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in \mathcal{B}} x_i \quad + \quad \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{B}, s \in \mathcal{S}} c_{is} z_{is} && && \text{(LP-NA)}\\
\text{subject to} \quad & \sum_{i \in \mathcal{B}} z_{is} \quad = \quad 1, && \forall s \in \mathcal{S} && (3.5)\\
& z_{is} \quad \leq \quad x_i, && \forall i \in \mathcal{B}, s \in \mathcal{S} &&\\
& x_i, z_{is} \quad \in \quad [0,1] && \forall i \in \mathcal{B}, s \in \mathcal{S} &&
\end{aligned}
$$

3.4 Competing with the Non-Adaptive Benchmark

In this section we give a summary of the attempts of Chawla et al. to compete against the optimal non-adaptive strategy. Recall that a non-adaptive strategy probes a fixed subset of boxes, and then picks a probed box of minimum cost. Is it possible to efficiently find an adaptive strategy that performs just as well? The authors show two results. On the one hand, they prove that we can efficiently find an SPA strategy that beats the performance of the optimal NA strategy. This along with Theorem 3.3.3 implies that we can efficiently find an $e/(e-1) \approx 1.582$-competitive PA strategy. First, we will describe their SPA strategy that is 1 competitive with the optimal non-adaptive cost.

Algorithm 3: SPA vs NA

Data: Solution $\boldsymbol{x}, \boldsymbol{z}$ to program (LP-NA); scenario s

1 $\sigma :=$ For $t \geq 1$, select and open box i with probability $\frac{x_i}{\sum_{i \in \mathcal{B}} x_i}$.

2 $\tau_s :=$ If box i is opened at step t, select the box and stop with probability $\frac{z_{is}}{x_i}$.

Lemma 3.4.1. *We can efficiently compute a scenario-aware partially-adaptive strategy with competitive ratio* 1 *against the optimal non-adaptive strategy.*

Putting this together with Theorem 3.3.3 we get the following theorem.

Theorem 3.4.2. *We can efficiently find a partially-adaptive strategy with total expected cost at most $e/(e-1)$ times the total cost of the optimal non-adaptive strategy.*

On the other hand, Chawla et al. show that it is NP-hard to obtain a competitive ratio better than 1.278 against the optimal NA strategy even using the full power of FA strategies.

3.5 Extension to feasibility constraints

General Feasibility Constraints. Apart from the classical version of the Pandora's Box problem, Chawla et al. study some extensions to where there are feasibility constraints that limit what or how many boxes we can choose. More specifically, let $\mathcal{F} \subseteq 2^{\mathcal{B}}$ denote the feasibility constraint. The goal is to probe boxes in some order and select a subset of the probed boxes that is feasible. Once again, we can describe an algorithm using the pair (σ, τ) where σ denotes the probing order and τ denotes the stopping time at which the algorithm stops and returns the cheapest feasible set found so far. The total cost of the algorithm then is the cost of the feasible set returned plus the stopping time. Note that the algorithm faces the same feasibility constraint in every scenario. In their work, Chawla et al. conseider two different kinds of feasibility constraints. In the first, the algorithm is required to select exactly k boxes for some $k \geq 1$. In the second, the algorithm is required to select a basis of a given matroid.

Selecting k items. In this setting, the feasibility constraint $\mathcal{F}$ is that the algorithm should pick k boxes and minimize the total cost and query time. Chawla et al., design a PA strategy that achieves an $O(1)$ competitive ratio. In the special case where the cost of the boxes are either 0 or ∞, the problem is the Generalized Min Sum Set Cover problem that was first introduced in [citation]. Their algorithm generalizes the constant competitive ratio achieved in [BGK10] to cases where the boxes have arbitrary values.

The Linear Programming relaxation of the optimal SPA strategy for the k items setting is the following:

$$
\begin{aligned}
\text{minimize} \quad & \frac{1}{|\mathcal{S}|} \sum_{s\in\mathcal{S},t\in\mathcal{T}} (1-y_{st}) \;+\; \frac{1}{|\mathcal{S}|} \sum_{i\in\mathcal{B},s\in\mathcal{S},t\in\mathcal{T}} c_{is} z_{ist} && && \text{(LP-}k\text{-cover)}\\
\text{subject to} \quad & \sum_{i\in\mathcal{B}} x_{it} && = 1, && \forall t \in \mathcal{T}\\
& \sum_{t\in\mathcal{T}} x_{it} && \leq 1, && \forall i \in \mathcal{B}\\
& z_{ist} && \leq x_{it}, && \forall s \in \mathcal{S}, i \in \mathcal{B}, t \in \mathcal{T}\\
& \sum_{t'\leq t, i\notin A} z_{ist'} && \geq (k-|A|) y_{st}, && \forall A \subseteq \mathcal{B}, s \in \mathcal{S}, t \in \mathcal{T}\\
& x_{it}, \quad z_{ist}, y_{st} && \in [0,1] && \forall s \in \mathcal{S}, i \in \mathcal{B}, t \in \mathcal{T}
\end{aligned}
$$

Using a solution of the above LP, the below SPA strategy achieves an $O(1)$ competitive ratio.

Algorithm 4: SPA vs PA, k-coverage

Data: Solution $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ to above LP, scenario s

$\sigma :=$ For each phase $\ell = 1, 2, \ldots$, open each box i independently with probability $q_{i\ell} = \min\left(\alpha \sum_{t \le 2^\ell} x_{it}, 1\right)$.

$\tau_s :=$

  Define $t_s^* = \max\{t : y_{st} \le 1/2\}$.

  if $2^\ell \ge t_s^*$ **then**

    For each opened box i, select it with probability $\min\left(\frac{\alpha \sum_{t \le 2^\ell} z_{ist}}{q_{i\ell}}, 1\right)$.

    Stop when we have selected k boxes in total.

end

Selecting a basis of a matroid. In this setting, $\mathcal{F}$ requires us to select a basis of a given matroid. More specifically, assuming that boxes have an underlying matroid structure we seek to find a base of size k with the minimum cost and the minimum query time.

The LP formulation is similar to the one for the k-coverage constraint, presented in the previous section. Let $r(A)$ for any set $A \subseteq \mathcal{B}$ denote the rank of this set. The constraints are the same except for last constraint that ensures we select no more than the rank of a set and that the elements that remain unselected are adequate for us to cover the remaining rank respectively.

$$
\begin{aligned}
\text{minimize} \quad & \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}, t \in \mathcal{T}} (1 - y_{st}) \;+\; \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{B}, s \in \mathcal{S}, t \in \mathcal{T}} c_{si} z_{ist} && && \text{(LP-matroid)} \\
\text{subject to} \quad & \sum_{i \in \mathcal{B}} x_{it} && = 1, && \forall t \in \mathcal{T} \\
& \sum_{t \in \mathcal{T}} x_{it} && \le 1, && \forall i \in \mathcal{B} \\
& \sum_{t \in \mathcal{T}, i \in A} z_{ist} && \le r(A), && \forall s \in \mathcal{S}, A \subseteq \mathcal{B} \\
& z_{ist} && \le x_{it}, && \forall s \in \mathcal{S}, i \in \mathcal{B}, t \in \mathcal{T} \\
& \sum_{i \notin A} \sum_{t' \le t} z_{ist'} && \ge (r([n]) - r(A)) y_{st}, && \forall A \subseteq \mathcal{B}, s \in \mathcal{S}, t \in \mathcal{T} \\
& x_{it}, && z_{ist}, y_{st} \in [0, 1] && \forall s \in \mathcal{S}, i \in \mathcal{B}, t \in \mathcal{T}
\end{aligned}
$$

Chawla et al. prove that the following SPA strategy is $O(\log k)$ competitive to the optimal partially-adaptive algorithm for picking a matroid basis of rank k.

Algorithm 5: SPA vs PA, matroid

Data: Solution $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ to above LP, scenario s

$\sigma :=$ for every $t = 1, \ldots, n$, open each box i independently with probability $q_{it} = \min\left\{\alpha \ln k \frac{\sum_{t' \leq t} x_{it'}}{t}, 1\right\}$.

$\tau_s :=$

  Let $t_s^* = \min\{t : y_{st} \leq 1/2\}$.

  if $t > t_s^*$ **then**

    For each opened box i, select it with probability $\min\left\{\frac{\alpha \ln k \sum_{t' \leq t} z_{ist'}}{t q_{it}},\ 1\right\}$.

    Stop when we find a base of the matroid.

end

Then, through the following Theorem, they argue that this competitive ratio is asymptotically tight for the case of the Partition Matroid.

Theorem 3.5.1. *Assuming NP⊈RP, no computationally efficient fully-adaptive algorithm can approximate the optimal non-adaptive cost within a factor of* $o(\log k)$.

This Lower Bound, however, does not hold for every matroid. For instance, the k items setting is captured by the uniform matroid where Chawla et al. achieved an $O(1)$ competitive ratio. It is interesting to examine whether there are other matroids for which the bound also holds.

Chapter 4

Graph Connectivity

4.1 Problem Definition

Graph connectivity is a fundamental combinatorial optimization problem and there are many real-world cases where it appears in a stochastic probing setting. Some real-world examples might be computer networks, gas pipelines, or perhaps electrical grids. In all the aforementioned cases, the network's nodes have to be connected in order for the system to be operational. However, connections might be unreliable or damaged and in most cases checking their reliability may be costly. For instance, in the case of a gas pipeline, a technician would have to physically visit the locations and perform certain diagnostic tests. In these examples, after an event (e.g. a natural disaster or a power outage) that has damaged an unknown part of the network, we would like to bring the network back to life by finding a spanning tree, while checking as few connections as possible.

In the past, researchers have studied $s - t$ connectivity ([FFX$^+$17]) and the minimum spanning tree problem ([GV04]) in a similar setting where they have modeled their graphs as ER random graphs ([ER59]), assuming that the existence probabilities of different edges are independent. This however does not always bear out in practice. In our work, we allow edge probabilities to be arbitrarily correlated and we assume that they are drawn from a joint distribution over graphs.

Model

In this problem, we are given a graph G, which we call *moldgraph*, and a distribution $\mathcal{D}$ over connected subgraphs of G. Nature chooses a graph G' from the distribution $\mathcal{D}$, whose edge set is initially hidden. We refer to G' as the *instantiated* or *realized* graph.

Algorithms are allowed to probe an edge of G in order to reveal whether it exists (i.e. belongs to G') or not. Their goal is to find a spanning tree of the realized graph G', while spending as little time as possible gathering information. Note that we assume that all potential subgraphs G' are connected, since completely discovering some connected components without knowing their exact number requires us to probe all the edges of G.

More formally, let $G' \sim \mathcal{D}$ be the instantiated subgraph of G and let $Q_{G'}$ be the random variable denoting the set of queried edges. Each valid $Q_{G'}$ has to contain at least one spanning

tree of G. Our goal then is to minimize the expected total probing time

$$\mathbf{E}_{G' \sim \mathcal{D}} [|Q_{G'}|]$$

Notice that our problem is captured by the matroid extension of the Pandora's Box problem, defined in [CGT+20], when we work with the Graphic Matroid. The $\Omega(\log k)$ lower bound on the approximation of Non-Adaptive strategies proved by Chawla et al. for the Partition Matroid, does not transfer to our case. It will be interesting to see whether a similar lower bound exists or whether adaptivity can provide better approximations.

4.2 Classes of Algorithms Based on Adaptivity

The optimal solution to the problem is an online algorithm that decides which edge to query next based on all the answers it has received on its previous queries. However, as mentioned in [CGT+20], these strategies are impossible to find or to approximate with samples. For instance, the existence or absence of some specific edges might encode the realized subgraph. While one can design strategies that find a spanning tree with a few queries, any reasonable approximation to the optimal cost would need to accurately learn this mapping. Learning such an arbitrary mapping is impossible through samples, unless there is significant probability of seeing the exact same combination of costs. As a result, using the class of all adaptive strategies as a benchmark will yield weak guarantees even for good strategies that are able to generalize to rich distributions. In order to provide meaningful guarantees, we define some natural classes of algorithms which we will use as benchmarks.

Definition 4.2.1 (Adaptivity of Strategies)**.** In a *Non-Adaptive (NA)* strategy, the set Q_G is fixed from the beginning. Algorithms in this family will query the edges in Q_G regardless of the instantiated graph and the information they gain by revealing edges.

A *Fully-Adaptive (FA)* strategy decides at each step which edge to query next, based on information gathered from previous steps. In other words, the set Q_G is not fixed from the beginning, but is determined in a dynamic fashion and depends on the instantiated graph.

4.2.1 Learning a good probing order

In this section, we will show that in order to optimize over all possible probing orders, it suffices to optimize with respect to a small set of instantiated graphs, drawn randomly from the distribution $\mathcal{D}$.

Let $\mathcal{D}$ be a distribution over connected subgraphs of the moldgraph G and $\mathcal{S}$ be a collection of m graphs drawn independently from $\mathcal{D}$. We will prove that for a permutation π of the moldgraph's edges, $cost_{\mathcal{D}}(\pi)$ will be close to $cost_{\mathcal{S}}(\pi)$ with high probability, where $cost_{\mathcal{D}}(\pi)$ is the expected cost of π on a graph drawn independently from the distribution $\mathcal{D}$ and $cost_{\mathcal{S}}(\pi)$ is the average cost of π on the sample set $\mathcal{S}$.

Lemma 4.2.2 (Lemma 3.3 of [CGT+20])**.** *Let $\epsilon, \delta > 0$ be given parameters. Let $\mathcal{S}$ be a set of m graphs chosen independently at random from $\mathcal{D}$ with $m = poly(n, 1/\epsilon, \log(1/\delta))$. Then, with*

probability at least $1-\delta$*, for all permutations* $\pi : [n] \to [n]$*, we have*

$$cost_{\mathcal{S}}(\pi) \in (1 \pm \epsilon) cost_{\mathcal{D}}(\pi).$$

Proof of Lemma 4.2.2. Fix a permutation π. For a given instantiated graph G' let $cost_{G'}(\pi)$ denote the total cost incurred by the permutation π on G'. Notice that $cost_{G'}(\pi) \in [n-1, m]$ where $n = |V(G)|$ and $m = |E(G)|$. Furthermore, let

$$cost_{\mathcal{D}}(\pi) = \mathbf{E}_{G' \sim \mathcal{D}} \left[cost_{G'}(\pi) \right]$$

be the expected cost of π on the distribution $\mathcal{D}$ and

$$cost_{\mathcal{S}}(\pi) = \frac{1}{|\mathcal{S}|} \sum_{G' \in \mathcal{S}} cost_{G'}(\pi)$$

be the average cost of π on the sample set $\mathcal{S}$. We denote the sample size $|\mathcal{S}|$ by k. Using Hoeffding's inequality, we get

$$\mathbf{Pr} \left[cost_{\mathcal{S}}(\pi) \geq (1 \pm \epsilon) cost_{\mathcal{D}}(\pi) \right] \leq 2e^{\frac{-2\epsilon^2}{k(m-n+1)^2}}$$

Taking a union-bound over all possible permutations, yields

$$\mathbf{Pr} \left[\cup_{\pi} [cost_{\mathcal{S}}(\pi) \geq (1 \pm \epsilon) cost_{\mathcal{D}}(\pi)] \right] \leq 2\ m!\ e^{\frac{-2\epsilon^2 k}{(m-n+1)^2}}$$

Requiring that the above probability is less than δ, gives us a sample complexity of

$$k \geq \frac{(m-n+1)^2}{2\epsilon^2} \left(m \log m + \log \left(\frac{1}{\delta} \right) \right)$$

□

4.2.2 LP Formulation of Non-Adaptive Strategies

We now construct the LP relaxation of the optimal Non-Adaptive cost. Recall that every algorithm has sample access to the distribution $\mathcal{D}$ and assume that a NA algorithm has drawn a set S of k samples from the distribution $\mathcal{D}$: $S = \{G_1, \ldots, G_k\}$ and goes on to decide the set Q that it will query on all future instances. Finding the minimum such set of edges, so that it contains a spanning tree on all the samples drawn is formulated by the below relaxed linear program.
For some subset $A \subseteq V(G)$ of the moldgraph's vertices and index $i \leq |S|$, we denote by $\delta_i(A)$ the set of edges of the sampled graph G_i that belong to the cut $(A, V(G) \setminus A)$

$$\text{minimize} \quad \sum_{e \in E(G)} x_e \tag{LP-NA}$$

$$\text{subject to} \sum_{e \in \delta_i(A)} x_e \geq 1 \qquad \forall i \in [k], A \subset V(G) \tag{4.1}$$

For an edge $e \in E(G)$, x_e denotes whether the edge will be included in the set Q. Constraint (4.1) guarantees that the algorithm will select at least one edge from every cut of every subgraph it has sampled. Note that there are exponentially many constraints, but we can solve the Linear Programming relaxation of this program since given a solution we can determine in polynomial time whether all sampled graphs are connected.

4.2.3 Finding the best Non-Adaptive Algorithm

In this section we provide a randomized rounding technique that returns a Non-Adaptive strategy that is a $O(2\log n + \log|\mathcal{D}|)$ approximation of the optimal Non-Adaptive cost, where $|\mathcal{D}|$ is the support of the input distribution $\mathcal{D}$. We then prove that the $O(\log|\mathcal{D}|)$ factor is optimal, by showing that no computationally eficient Non-Adaptive algorithm can approximate the optimal Non-Adaptive cost within a factor that is less than logarithmic in the size of the support of the distribution $\mathcal{D}$.

Algorithm 6: A Non-Adaptive strategy via Randomized Rounding

Input: x solution of LP-NA
for $i = 1$ ***to*** $2\log n + \log|\mathcal{D}|$ **do**
  for $e \in E(G)$ **do**
    Select edge e with probability x_e
  end
end

Lemma 4.2.3. *Algorithm 6 returns a Non-Adaptive strategy that satisfies all constraints with high probability and on expectation it is a $O(2\log n + \log|\mathcal{D}|)$ approximation of the optimal Non-Adaptive cost.*

Proof of Lemma 4.2.3. We will prove that the edges selected contain a spanning tree for each sampled graph with high probability.

The probability that a given cut C remains uncovered after one round of selections is

$$\prod_{e \in \delta_i(C)} (1 - x_e) = \prod_{e \in \delta_i(C)} e^{-x_e} = e^{-\sum_{e \in \delta_i(C)} x_e} \leq \frac{1}{e}$$

The probability that a given cut C will remain uncovered after all the rounds is at most

$$\left(\frac{1}{e}\right)^{2\log n + \log|\mathcal{D}|} \leq \frac{1}{n^2|\mathcal{D}|}$$

Taking a union bound on all sampled graphs and n independent cuts of each graph, we have that the probability that there exists an uncovered cut in any sampled graph is at most

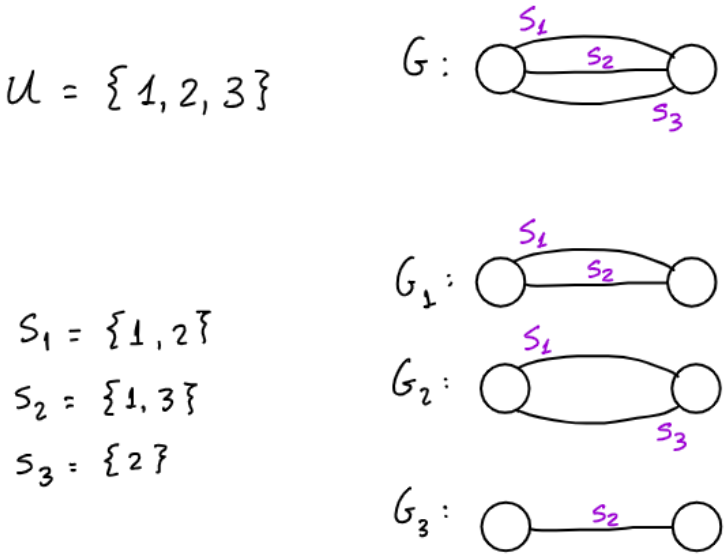
$$\sum_{i \in |dist|} \sum_{C \in \delta(G)} \frac{1}{n^2 |\mathcal{D}|} = n|\mathcal{D}| \cdot \frac{1}{n^2 |\mathcal{D}|} = \frac{1}{n}$$

□

Lemma 4.2.4. *Assuming $P \neq NP$, no computationally efficient Non-Adaptive algorithm can approximate the optimal Non-Adaptive cost within a factor of $O(\log |\mathcal{D}|)$, where $|\mathcal{D}|$ is the size of the support of the distribution $\mathcal{D}$.*

Proof of Lemma 4.2.4. We will show this by reducing Set Cover to our problem. Suppose we are given a set of elements $\mathcal{U} = \{1, \ldots, n\}$ and a collection of subsets of $\mathcal{U}$: $S_1, \ldots, S_m$ whose union equal the universe. We are asked to identify the smallest sub-collection of $\mathcal{S}$ whose union equals the universe.

We form an instance of our problem, selecting as moldgraph two vertices with m parallel edges. The i-th set of $\mathcal{S}$, S_i, corresponds to the $i-th$ edge connecting the two vertices. Furthermore, for each element of the universe $j \in \mathcal{U}$ we will create a subgraph G_j of G that consists only of the edges that correspond to the sets that contain j, as shown in the following figure. We choose the distribution $\mathcal{D}$ to be the uniform distribution over the subgraphs G_i.



Any Non-Adaptive algorithm will try to select a fixed set Q of edges that contains a spanning tree for each sample. Notice that any fixed set Q is a solution for the Set Cover instance and the cost of the optimal Non-Adaptive solution is equal to the cost of the optimal solution of the given Set Cover instance. From [LY94], we know that we can not approximate the optimal solution of Set Cover within a factor of $O(\log n) = O(\log |D|)$, unless $P = NP$. □

4.3 Deriving Hardness from the Oracle Setting with One-Sided Error

In this section we introduce a new problem that is provably a bottleneck case of our original problem (henceforth basic model). We can use this new setting to prove competitive ratio lower bounds for the basic model and also to design algorithms and later translate them to algorithms for the basic model.

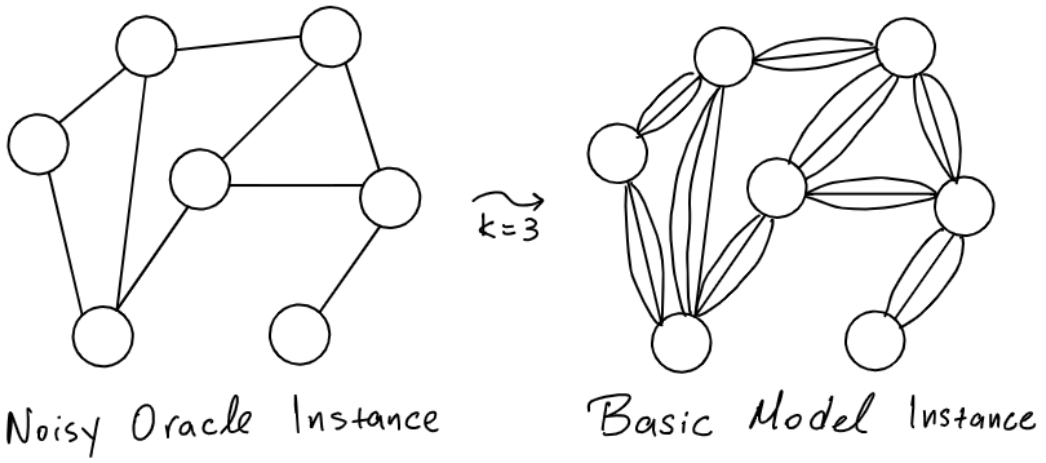
4.3.1 Definition and reduction to the Basic Model

In the *Noisy Oracle* setting we we are given a graph G and a distribution $\mathcal{D}$ over connected subgraphs of G. Nature chooses a graph G' from the distribution $\mathcal{D}$. The algorithms can ask a noisy oracle about whether an edge exists or not. The oracle works as follows; if the queried edge exists then it answers "Yes" with probability p and "No" with probability $1-p$. If the queried edge is not realized, ie it does not belong to $E(G')$, then it answers "No" with probability 1. In other words, if we get a positive answer we are sure that the edge exists, but when we get a negative answer we can not know whether this is a false negative or not. The goal of the algorithm is then to minimize the expected number of queries until we reveal a spanning tree of G'.

This is a similar and much more convenient setting to design algorithms in. But the motivation behind its definition stems from its ability to provide lower bounds for the basic model, as it is formally described in the next theorem.

Theorem 4.3.1. *Suppose that no strategy can find a spanning tree in the noisy setting while performing less than $\frac{c}{p} \cdot m$ queries on expectation, where $m = |E(G)|$. Then, in the basic model, there exists no computationally efficient Fully-Adaptive algorithm that can approximate the optimal Non-Adaptive cost within a factor that is less than c.*

Proof of Theorem 4.3.1. Let G_N be the moldgraph and $\mathcal{D}_N$ be the input distribution for the noisy oracle problem. We construct an instance of the basic model whose moldgraph G_B is created by replacing each edge of G_N by k parallel edges, as shown in the following figure, where k is a constant that will be determined later.



For every graph G'_N that has non-zero probability p_N under the distribution $\mathcal{D}_N$, we will construct several graphs for the distribution $\mathcal{D}_B$ as follows:

- If an edge e of G_N is not realized in G'_N then all k corresponding parallel edges will not exist.
- If an edge e is realized, then we create all the $\binom{k}{pk}$ subsets where pk edges are realized.

In this way, for each graph G'_N we will construct at most $\binom{k}{pk}^m$ graphs for the distribution $\mathcal{D}_B$ and we will evenly split the probability p_N among these graphs.

Notice that with this construction, no matter which parallel edges have been revealed, the remaining ones will always have the same probability of existence. Therefore, any Fully Adaptive algorithm can do no better than select one from the remaining edges uniformly at random.

As a result, we can simulate every strategy in the basic model and create an algorithm for the noisy oracle setting. When the basic model algorithm queries an edge of a set of parallel edges, we make a query in the edge corresponding to that set in the noisy oracle setting. On expectation, the basic model algorithm will need $1/p$ queries to find an edge in the set of parallel edges, as will the noisy oracle strategy. Therefore, for every Fully Adaptive Algorithm ALG_{FA} for the basic model, there exists a that solves the noisy setting with the same expected number of queries, that is $ALG_{FA} = ALG_{Noisy}$.

Notice that selecting the first $pk+1$ edges from every set of parallel edges, is a Non-Adaptive strategy that is guaranteed that it will find an instantiated spanning tree. Therefore, for the optimal Non-Adaptive cost it holds that

$$OPT_{NA} \leq m(pk+1)$$

Choosing k to be $\frac{1-p}{p^2}$ and using the previous equality we get that

$$\frac{ALG_{FA}}{OPT_{NA}} \geq \frac{ALG_{Noisy}}{m/p}$$

If there exists a lower bound in the form of $ALG_{Noisy} \geq \frac{c}{p}m$, then we know that there does not exist a Fully Adaptive Algorithm that can approximate the optimal Non-Adaptive cost within a factor that is less than c, that is

$$\frac{ALG_{FA}}{OPT_{NA}} \geq c$$

□

4.3.2 Formulation as a Markov Decision Process

Markov Decision Processes (MDPs) are discrete-time stochastic control processes that were introduced by Bellman in [BEL57]. They provide a mathematical framework to formulate decision-making problems where outcomes are partly random and partly under the control of a decision-maker. Some great resources on MDPs are [How60] and Chapter 3 of [SB18]

The noisy setting can be formulated as a Markov Decision Process with the following characteristics:

State Space: the state s is an m-dimensional vector with each entity taking values in $\{-1, \ldots, \log n\}$. Each dimension i of s corresponds to an edge and $s[i]$ indicates whether this edge has been discovered as realized (in which case $s[i]$ will be -1) or the total number of negative queries performed on this edge by the algorithm. The state space is exponential in the size of the input, as $|\mathcal{S}| = m^{\log n+1}$. As a result, the common dynamic programming techniques that are used to find the optimal policy in finite horizon MDPs, will have exponential complexity.

Action Set: in each state the algorithm decides which edge to query next. The set of actions is, therefore, the set $E(G)$ of the moldgraph's edges.

Rewards: for a state $s \in \mathcal{S}$, the reward $R(s)$ will be 1 if the discovered edges span the graph and 0 otherwise.

Transition Probabilities: The transition probability from a state $s \in mathcalS$ while querying an edge $e \in E(G)$ depend on the existence probability of e. If e corresponds to the i-th dimension of the state s, then the only possible next states s' are those that identical with s in all dimensions except $s[i]$. The next states can either have $s'[i] = -1$ if e is discovered, or $s'[i] = s[i] + 1$.

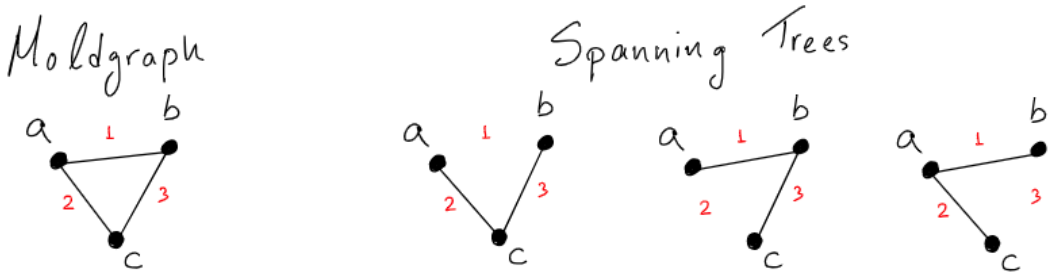
The MDP formulation of the noisy setting gives us a way to compute the optimal policy for certain inputs. Of course, as we mentioned earlier, all the dynamic programming methods known will have exponential complexity, but we can use them to solve small instances and derive lower bounds. Combining this technique with Theorem 4.3.1 proves inaproximability results for Fully Adaptive algorithms in the basic model.

Theorem 4.3.2. *No computationally efficient fully-adaptive algorithm can approximate the optimal non-adaptive strategy within a factor that is less than* 1.55.

To prove Theorem 4.3.2, we will use the following Lemma.

Lemma 4.3.3. *The optimal fully adaptive cost for the noisy setting with $p = 1/2$, where the moldgraph is the K_3 graph and the distribution $\mathcal{D}$ is the uniform over its 3 connected subgraphs, is 9.323 queries on expectation.*

Proof of Lemma 4.3.3. We are given K_3 as the moldgraph and the input distribution to be the uniform over its 3 connected subgraphs. We will formulate the problem as a Markov Decision Process. The state s will be a 3-dimensional vector with one entity for each edge (we number the edges as seen below).



In order to calculate the transition probabilities, we will need the probability that each edge exists given the queries that have been made. The probability that an edge exists is simply the summation of the probability that a spanning tree is realized for the trees that contain the given edge:

$$\mathbf{Pr}\left[e \text{ exists}\right] = \sum_{T \in \mathcal{D}:\ e \in T} \mathbf{Pr}\left[T \text{ is realized}\right]$$

The probability that each tree is realized begins at $1/3$ since $\mathcal{D}$ is the uniform distribution. However, with each negative answer we have to do a bayesian update on the existence probabilities of all trees.

Let $\mathbf{P}_t\left(T_i\right)$ be the probability that T_i exists in round t, that is given all the answers in the previous $t-1$ rounds. Also, let $\mathbf{P}_t\left(N_e\right)$ denote the probability that we will get a negative answer on edge e on round t.

It holds that

$$\mathbf{P}_{t+1}\left(T_i\right) = \mathbf{P}_t\left(T_i | N_{et}\right) = \frac{\mathbf{P}_t\left(N_{et}|T_i\right)\mathbf{P}_t\left(T_i\right)}{\mathbf{P}_t\left(N_{et}\right)} = \frac{\mathbf{P}_t\left(N_{et}|T_i\right)\mathbf{P}_t\left(T_i\right)}{C_t} =$$

where

$$C_t = \mathbf{P}_t\left(N_{et}\right) = \sum_{T \in \mathcal{D}} \mathbf{P}_t\left(N_{et}|T_i\right) \cdot \mathbf{P}_t\left(T_i\right)$$

Suppose that we get a negative answer on edge e, the updated tree probabilities are:

$$e \in T_i \Rightarrow \mathbf{P}_{t+1}(T_i) = \frac{1}{C_t} \cdot \frac{\mathbf{P}_t(T_i)}{2}$$

$$e \notin T_i \Rightarrow \mathbf{P}_{t+1}(T_i) = \frac{1}{C_t} \cdot \mathbf{P}_t(T_i)$$

The updated probability that edge e exists is

$$\mathbf{P}_{t+1}(e) = \frac{\sum_{T_i \in \mathcal{D}:\ e \in T_i} \mathbf{P}_{t+1}(T_i)}{\sum_{T_i \in \mathcal{D}} \mathbf{P}_{t+1}(T_i)} = \frac{1}{2} \cdot \frac{\sum_{T_i \in \mathcal{D}:\ e \in T_i} \mathbf{P}_t(T_i)}{\sum_{T_i \in \mathcal{D}} \mathbf{P}_t(T_i)} = \frac{1}{2} \cdot \mathbf{P}_t(e)$$

Using the above bayesian updates and the Value Iteration Algorithm (first introduced in [BEL57]), we can compute the optimal policy and its expected cost which is 9.323 □

We can now proceed to prove Theorem 4.3.2

Proof of Theorem 4.3.2. From Lemma 4.3.3 it holds that there exists no strategy that can achieve an expected cost of $1.55 \cdot 2m$ on every instance. As a result, from Theorem 4.3.1, there exists no computational fully-adadptive algorithm that can approximate the optimal non-adaptive cost within a factor of 1.55. □

4.4 Competing with the Non-Adaptive Benchmark

Following up on 4.2.4, it is natural to question whether adaptivity can yield better approximations of the optimal Non-Adaptive cost. In this section we will use the samples drawn from the distribution $\mathcal{D}$ to design Adaptive Algorithms with good guarantees. We will prove that, depending on characteristics of the graph G, they can achieve a constant competitive ratio against the optimal Non-Adaptive strategy with respect to the distribution $\mathcal{D}$.

4.4.1 Separating Cut Selection and Querying a Cut

The only guarantee we have in this problem is that all cuts of the moldgraph will contain at least one instantiated edge. The algorithms that we propose in this section are exploiting this constraint and are focusing on one cut at a time, trying to reveal at least one edge. All such algorithms can be viewed as a framework consisting of two parts:

1. A subroutine that selects the next cut to cover given all the queries that have been made.
2. A subroutine that queries a given cut until it discovers an edge.

The advantage of this design choice is that all algorithms for the noisy setting can be easily translated to algorithms for the basic problem. The subroutine that selects the next cut to query remains the same and we only have to change our cut querying subroutine.

We will now describe a cut-querying subroutine for each setting. Fix a moment in the execution of any algorithm and let $A_1, \ldots, A_{k+1}$ be the connected components that have been formed

(initially each component will be a single vertex). Our subroutine will examine cuts between a single connected component and all the others. Let $S \subset E(G)$ be the cut that separates A_{k+1} from the other components. We can write S as a union of disjoint edge sets $B_1 \cup \cdots \cup B_k$ where

$$B_i = \{(u, v) \in E(G) : u \in A_i \text{ and } v \in A_{k+1}\}$$

We know that S has at least one instantiated edge. Upon finding an edge e that belongs in a set B_j, the connected components A_{k+1} and A_j will be merged. Note that any rational adaptive algorithm will never again query any edge that belongs to B_j. Therefore, for our subroutine to be efficient, we would like the total amount of queries performed to be proportional to the size of the edge set that will be contracted and never queried again.

First, let's describe such a subroutine for the noisy oracle setting. The subroutine will be given a cut $S \subset E(G)$, divided in k disjoint edge sets $B_1, \ldots, B_k$ and will return an instantiated edge $e \in S$.

Algorithm 7: $DISCOVER_1(S)$: Cut Querying Subroutine for the Noisy Oracle Setting

Input: Set $\mathcal{S} = B_1 \cup B_2 \cup \cdots \cup B_k$

```
1  ∀_{i∈{1,...,k}} next_edge[i] = 0;
2  while no edge is found do
3      for i = 1 to k do
4          Query next_edge[i] of B_i;
5          if edge exists then
6              return (next_edge[i], i);
7          end
8          next_edge[i] = (next_edge[i] + 1)  mod |B_i|;
9      end
10 end
```

Theorem 4.4.1. *Algorithm 7 finds an edge e of some component B_j performing at most $2k|B_j|$ queries on average.*

Proof of Theorem 4.4.1. Without loss of generality, let's assume that the edge set B_j contains an instantiated edge of this cut. On average, the oracle will return a positive answer on the instantiated edge after 2 queries on expectation. Since we are querying all the edges of B_j in a circular manner, we will have performed at most $2|B_j|$ queries on average, until we find the instantiated edge. In the meantime, the subroutine was also performing queries on the remaining $k - 1$ edge sets. Since it does queries on parallel on all the edge sets, we will have performed exactly $2|B_j|$ queries on every other edge set as well. Therefore, in total the subroutine will conduct at most $2k|B_j|$ queries on average.

□

Algorithm 8: Cut Querying Subroutine for the Basic Model

Input: Set $S = B_1 \cup \cdots \cup B_k$ of edges, x solution of LP

$\text{OPT}_i = \sum_{e \in B_i} x_e$

while *no edge is found* **do**

  for $i = 1$ ***to*** k **do**

    Select and query edge $e \in B_i$ with probability $\frac{x_e}{\text{OPT}_i}$

  end

end

Theorem 4.4.2. *Algorithm 8 finds an edge e of some component B_j performing on average at most $k^2 OPT_i$ queries.*

To prove this theorem, we use the following lemma.

Lemma 4.4.3. *The expected number of queries that Algorithm 8 performs on a single edge set B_j that contains an instantiated edge, is at most OPT_j/δ where $OPT_j = \sum_{e \in B_j} x_e$ and δ is the sum of x_e of the instantiated edges of B_j.*

Proof of Lemma 4.4.3. Let $C_j \subseteq B_j$ be the subset of edges of B_j that are instantiated. We have assumed that $C_j \neq \emptyset$. The cost that the fractional optimal solution incurs on B_j is

$$\text{OPT}_j = \sum_{e \in B_j} x_e$$

Let

$$\delta = \sum_{e \in C_j} x_e$$

The algorithm is sampling edges, each with probability x_e/OPT_j. The probability that it samples an edge from C_j is

$$P[\text{sample instantiated edge}] = \frac{\sum_{e \in C_j} x_e}{\text{OPT}_j} = \frac{\delta}{\text{OPT}_j}$$

The number of samples needed until it samples a realized edge, is a random variable following a Geometric distribution with $p = \delta/\text{OPT}_j$. Hence, the expected number of queries is

$$\frac{1}{P[\text{sample instantiated edge}]} = \frac{\text{OPT}_j}{\delta}$$

□

Proof of Theorem 4.4.2. Let $C \subseteq S$ be the realized edges of the cut S. The optimal Non-Adaptive solution will have to select at least one realized edge in this cut, therefore

$$\sum_{e \in C} x_e \geq 1$$

We can easily observe that there will be at least one of the k edge sets, let it be B_j, where the sum of x_e of the realized edges it contains will be at least $1/k$. Using Lemma 4.4.3, Algorithm 8 will find a realized edge in B_j after $\text{OPT}_j/(1/k) = k\text{OPT}_j$ queries. Since it performs one query on each edge set in every iteration, it will have asked exactly $k\text{OPT}_j$ queries on every other edge set as well. Therefore, the total number of queries will be $k^2\text{OPT}_j$ on average. □

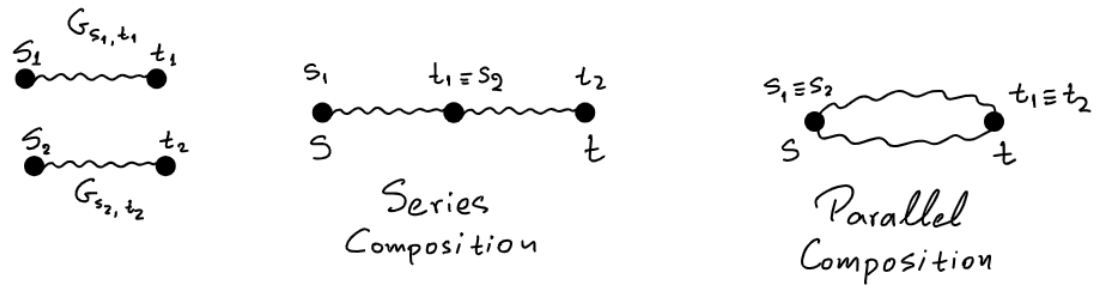
4.4.2 Warm-Up Result: Series Parallel Graphs

We will now focus on designing subroutines that select the next cut to query given all the realized edges that have been revealed. These strategies, coupled with the Cut Querying Subroutines 8 and 7, yield adaptive algorithms for both settings.

As a warm-up for our main result, we will first design an algorithm for Series-Parallel Graphs. More specifically, we will restrict the moldgraph to be a Series-Parallel Graph and taking advantage of its structure and recursive definition, we will design an algorithm that approximates the optimal Non-Adaptive cost within a constant. First, let's define the Series-Parallel Graphs.

Definition 4.4.4 (Series-Parallel Graph)**.** A graph $G_{s,t}$ with a source s and a sink t is a series-parallel (henceforth SP) graph if one of the following conditions hold

- $G_{s,t}$ consists only of vertices s and t connected with an edge
- $G_{s,t}$ can be created by *Parallel Composition* of two SP graphs G_{s_1,t_1} and G_{s_2,t_2}, that is merging s_1 with s_2 and t_1 with t_2.
- $G_{s,t}$ can be created by *Series Composition* of two SP graphs G_{s_1,t_1} and G_{s_2,t_2}, that is merging t_1 with s_2.



Working with the recursive definition, a naive approach would be to find a spanning tree for each of the SP components of G recursively and then merge the results. However, notice that when G is the parallel composition of two SP graphs G_1 and G_2, it is not necessary that both

subgraphs will be connected using realized edges. For instance, G_1 might have an $s-t$ cut and its vertices might be connected through G_2.

Although, with a more careful analysis we can observe that each SP component will have at most one cut. In other words, it will be possible to connect its vertices into at most two connected components.

Definition 4.4.5 (Up-to-1 Connectivity)**.** A graph G is up-to-1 connected if it has at most two connected components.

We will now describe an algorithm that works recursively on series-parallel graphs and uses the cut querying subroutines of Section 4.4.1 to make an SP graph up-to-1 connected.

Algorithm 9: SolveSP($G_{s,t}$): makes the SP Graph $G_{s,t}$ Up-To-1 Connected

```
if V(G_{s,t}) == {s,t} then
    return ∅
end
Let G_l, G_r be two SP components of G_{s,t}
Q ← SolveSP(G_l) ∪ SolveSP(G_r)
if G_{s,t} is a Series Composition of G_l and G_r then
    Let C_l, C_r be the uncovered s − t cuts of G_l and G_r
    S ← {C_l, C_r}
    Q ← Q ∪ DISCOVER(S)
end
return Q
```

Lemma 4.4.6. *Using Algorithm 9 and Cut Querying Subroutine 7 one can find a spanning tree of a Series-Parallel graph G, performing at most $4|E(G)|$ queries on expectation.*

Proof of Lemma 4.4.6. We will prove by induction on the number of edges that when Algorithm 9 is called on a series-parallel graph G, it returns two connected components that are separated by a cut C, while performing on expectation at most $4(|E(G)| - |C|)$ queries.

The proposition holds for the base case where graph G has exactly two vertices connected with an edge. Suppose that it holds for SP Graphs with at most $m-1$ edges. Let G be an SP

graph with m edges. The graph G will be the composition, either series or parallel, of two SP graphs G_1 and G_2. We run Algorithm 9 on G_1 and G_2 and by the induction hypothesis we get two connected components C_{11}, C_{12} and C_{21}, C_{22} for each graph. Let S_1 and S_2 be the uncovered cuts respectively. We consider the following cases:

1. If G is the parallel composition of G_1 and G_2, then with no further queries Algorithm 9 returns the connected components $C_1 = C_{11} \cup C_{21}$ and $C_2 = C_{12} \cup C_{22}$. In total it will have performed at most

$$\begin{aligned} Q(G) &= Q(G_1) + Q(G_2) \\ &= 4(|E(G_1)| - |S_1|) + 4(|E(G_2)| - |S_2|) \\ &= 4(|E(G_1)| + |E(G_2)| - |S_1| - |S_2|) \\ &= 4(|E(G)| - |S|) \end{aligned}$$

   queries on expectation.

2. If G is the series composition of G_1 and G_2, the algorithm will run the cut querying subroutine on $S = S_1 \cup S_2$. Without loss of generality, assume that it finds a realized edge in S_1. By Theorem 4.4.1, it will have performed at most $4|S_1|$ queries. The algorithm then returns the connected components $C_1 = C_{11} \cup C_{12} \cup C_{21}$ and $C_2 = C_{22}$ having performed at most

$$\begin{aligned} Q(G) &= Q(G_1) + Q(G_2) + 4|S_1| \\ &= 4(|E(G_1)| - |S_1|) + 4(|E(G_2)| - |S_2|) + 4|S_1| \\ &= 4(|E(G_1)| + |E(G_2)| - |S_2|) \\ &= 4(|E(G)| - |S|) \end{aligned}$$

   queries on expectation, which concludes the proof

□

4.4.3 An Algorithm Based on the Moldgraph's Treewidth

In this section, we present an adaptive strategy that selects the next cut to query by utilizing a tree decomposition of the moldgraph. Combining this framework with the Cut Querying subroutine described in Algorithm 7, yields an Algorithm that solves the noisy problem while performing $O(km)$ queries on expectation, where k is the moldgraph's Treewidth.

Section A of the appendix contains a brief introduction to tree decompositions as well as some definitions that are necessary for this chapter. Lemma A.3.2, indicates that it suffices to design our algorithm on a "Nice" Tree Decomposition.

Suppose we are given a "Nice" Tree Decomposition T of the moldgraph G. We choose an arbitrary node $u \in T$ to be the root of the tree decomposition. We will use the following notation:

i) For a node $x \in T$, T_x denotes the subtree of the tree decomposition that has x as its root.

ii) $V(T_x)$ is the set of all the vertices of G that are present in at least one bag of at least one node of T_x. Formally, $V(T_x) = \{u \in V(G) : \exists a \in T_x \text{ s.t. } u \in X_a\}$

iii) $E(T_x)$ is the set of all of the moldgraph's edges that are between vertices in $V(T_x)$. Formally, $E(T_x) = \{(u, v) \in E(G) : u \in V(T_x) \text{ and } v \in V(T_x)\}$

We first show that for every subtree T_v of the moldgraph's tree decomposition, it is possible to connect all the vertices in $E(T_v)$ in up to $|X_v|$ connected components, using only instantiated edges from $E(T_v)$. This is described in the following lemma.

Lemma 4.4.7. *For every subtree T_v of the moldgraph's tree decomposition, one can form a forest of up to $|X_v|$ connected components that contain all the vertices in $V(T_v)$, using only instantiated edges from $E(T_v)$. Furthermore, each connected component C_i, will have at least one vertex in X_v, that is $\forall i \leq |X_v| : \ C_i \cap X_u \neq \emptyset$.*

The proof of Lemma 4.4.7 is deferred to Section B of the appendix.

Using Lemma 4.4.7 and the Cut Querying subroutines from Section 4.4.1, we will now describe an algorithm that works recursively on the subtrees of a tree decomposition with root u, and connects the vertices $V(T_u)$ into at most $|X_u|$ connected components. Note that we can add Forget Nodes as predecessors of the root u, removing all the vertices of X_u one by one until we are left with only one vertex. In that case, the algorithm will return an instantiated spanning tree of G.

Algorithm 10: CONNECT(T_u): Connects vertices into at most $|X_u|$ components

Input: T_u: "Nice" Tree Decomposition rooted at u
Output: Set of most $|X_u|$ connected components

if u *is a Leaf* **then**
  return $\emptyset$
else if u *is an Introduce Node* **then**
  Let u' be the child of u and $a = X_u \setminus X_{u'}$
  return CONNECT($T_{u'}$) $\cup$ { $\{a\}$ }
else if u *is a Join Node* **then**
  Let $u_1,\ u_2$ be the children of u
  return CONNECT(u_1) $\cup$ CONNECT(u_2)
else if u *is a Forget Node* **then**
  Let u' be the child of u and $a = X_{u'} \setminus X_u$
  $\{C_1, \dots C_{k+1}\}$ = CONNECT($T_{u'}$)
  Suppose $a \in C_{k+1}$
  Let $B_i = \{(u, v) \in E(T_u) : u \in C_i$ and $v \in C_{k+1}\}$
  DISCOVER($B_1 \cup \dots \cup B_k$)
  return $\{C_1, \dots C_k\}$

Theorem 4.4.8. *Algorithm 10 is $O(k^2)$ competitive against the optimal Non-Adaptive strategy, where k is the moldgraph's treewidth.*

To prove Theorem 4.4.8, we will use the following Lemma. Suppose that we are combining Algorithm 10 with a Cut Querying Subroutine that performs $l \cdot |B_j|$ queries when the contracted edge set is B_j.

Lemma 4.4.9. *When Algorithm 10 is called on a tree decomposition with root u, it returns at most $|X_u|$ connected components after performing $l \cdot (|E(T_u)| - |S|)$ queries on expectation, where S is the set of all the edges between different connected components.*

Proof of Lemma 4.4.9. We will prove Lemma 4.4.9 by induction on the subtrees of the tree decomposition. The base case of the induction is the root u of the tree decomposition being a leaf with $|X_u| = 1$. In this case, the lemma trivially holds.

Suppose that the lemma holds for the subtrees rooted at the children of the root u, we will show that it also holds for T_u. Consider the following cases:

1. If node u is an *Introduce Node*, then it will have exactly one child v with $X_u = X_v \cup \{a\}$ for some vertex $a \in G$ for which $a \notin X_v$. From the induction hypothesis on T_v, the Algorithm finds $|X_v|$ connected components performing $Q(T_v) = l \cdot (|E(T_v)| - |S_v|)$ queries on expectation. The algorithm then adds a new connected component containing only the vertex a and performs no additional queries. The total number of queries performed can be written as

$$\begin{aligned} Q(T_v) &= l \cdot (|E(T_v)| - |S_v|) \\ &= l \cdot (|E(T_v)| + |N_a| - |S_v| - |N_a|) \\ &= l \cdot (|E(T_u)| - |S_u|) \\ &= Q(T_u) \end{aligned}$$

where N_a is the set of edges adjacent to a, $N_a = \{(a, x) \in E(T_u) \text{ s.t. } x \in V(T_u)\}$

2. If node u is a *Forget Node*, then it will have exactly one child v with $X_u = X_v \setminus \{a\}$ for some vertex $a \in G$. From the induction hypothesis on T_v, the Algorithm finds $|X_v|$ connected components. It then proceeds to query the cut between the component containing a and all the others, using a Cut Querying Subroutine. Let $|B_j|$ be the subset of the cut S_v that will be contracted. The Cut Querying Subroutine will find the instantiated edge performing $l \cdot |B_j|$ queries on expectation. The total number of queries will be

$$\begin{aligned} Q(T_u) &= Q(T_v) + l \cdot |B_j| \\ &= l \cdot (|E(T_v)| - |S_v|) + l \cdot |B_j| \\ &= l \cdot (|E(T_u)| - (|S_u| + |B_j|)) + l \cdot |B_j| \\ &= l \cdot (|E(T_u)| - |S_u|) \end{aligned}$$

3. If node u is a *Join Node*, then it will have exactly two children u_1 and u_2 with $X_u = X_{u_1} = X_{u_2}$. We denote by B the set of edges between the vertices of the bag X_u, that is $B = \{(x, y) \in E(G) : x \in X_u \text{ and } y \in X_u\}$. Note that $|E(T_u)| = |E(T_{u_1})| + |E(T_{u_2})| - |B|$. We recursively run the algorithm on T_{u_1} providing it with all the edges in $E(T_{u_1})$ and then recursively call it on T_{u_2} passing only the edges $E(T_{u_2}) \setminus B$. From the induction hypothesis on the two children, we get

$$\begin{aligned} Q(T_u) &= Q(T_{u_1}) + Q(T_{u_2}) \\ &= l \cdot (|E(T_{u_1})| - |S_{u_1}|) + l \cdot (|E(T_{u_2})| - |B| - |S_{u_1}|) \\ &= l \cdot (|E(T_u)| - |S_u|) \end{aligned}$$

which concludes the proof.

□

We can now proceed to prove Theorem 4.4.8.

Proof of Theorem 4.4.8. Algorithm 10 will use Algorithm 8 as a Cut Querying Subroutine. From Theorem 4.4.2, we know that given a cut, Algorithm 8 performs $k^2 OPT_j$ queries, where

$$OPT_j = \sum_{e \in B_j} x_e$$

with B_j being the edge set that will be contracted. Lemma 4.4.9 still holds when the measure $|C|$ of a set of edges C, is defined as $|C| = \sum_{e \in C} x_e$ instead of the number of elements of the set. We add some Forget Nodes as predecessors of the root u of the given tree decomposition, removing all the elements of X_u one by one until we are left with a single vertex. Calling Algorithm 10 from the new root, will yield an instantiated spanning tree of G while the algorithm will have performed $k^2 \cdot OPT$ queries, where

$$OPT = \sum_{e \in E(G)} x_e$$

We know that $OPT_{NA} \geq OPT$, therefore Algorithm 10 is $O(k^2)$ competitive against the optimal Non-Adaptive Algorithm.

□

4.4.4 An Algorithm for Sparse and Minor-Closed Graphs

In this section, we study families of Graphs that are sparse, under a specific definition, and are also Minor-Closed. This means that after contracting found edges and merging all potential parallel edges, the sparsity properties still hold.

Definition 4.4.10. A graph G of n vertices and m edges, is called *ρ-Sparse* if and only if $m \leq \rho n$.

We provide an Algorithm that is $O(\rho)$ competitive for ρ-Sparse and Minor-Closed graphs. Although these conditions might seem strict, there are many such graph families, some of which also have unbounded treewidth. For instance, planar graphs have these properties for $\rho = 3$.

Algorithm 11: SolveSparse(G): Solves the problem in ρ-sparse minor-closed graphs.

```
if G is a single node then
    return ∅
end
Let u be the vertex with minimum degree in G.
e = DISCOVER(N(u))
Let G' be the resulting graph after contracting edge e.
return SolveSparse(G') ∪ {e}
```

Theorem 4.4.11. *Algorithm 11 performs at most $O(\rho m)$ queries on expectation*

Proof of Theorem 4.4.11. We will prove this by induction on the number n of vertices of G that Algorithm 11 performs at most $4\rho \cdot m$ queries on expectation in the noisy setting.

For the base case of $n = 1$ it holds trivially. Now assume that it holds for all graphs of at most $n - 1$ vertices. We will prove that it also holds for graphs of n vertices.

It is not hard to see that due to the ρ-sparsity of the graph, there always exists at least one vertex u with degree $d \leq 2\rho$. Otherwise, the number of edges would be equal to

$$\frac{1}{2}\sum_{v \in V} \deg(v) > \rho n$$

which is greater than $m = \rho n$.

The algorithm uses Algorithm 7 as a Cut Querying Subroutine passing the cut of the least degree vertex with the remaining graph. The Cut Querying Subroutine will find an instantiated edge inside a super-edge B_j, spending on expectation at most $4\rho \cdot |B_j|$ queries.
After contracting the super-edge B_j that contains the instantiated edge found by the subroutine, the algorithm will continue on the contracted graph that has $n - 1$ vertices. By the induction hypothesis, the algorithm will find an instantiated spanning tree performing at most

$$4\rho \cdot (|E(G)| - |B_j|)$$

queries on expectation. Therefore, the total number of queries performed will be

$$4\rho \cdot (|E(G)| - |B_j|) + 4\rho \cdot |B_j| = 4\rho \cdot |E(G)|$$

□

4.5 Concluding Remarks & Future Work

In this thesis we formulated the connectivity problem under a data-driven perspective and we defined the Non-Adaptive & Adaptive classes of Algorithms for this problem. We proved that it is NP-hard to find a computationally efficient Non-Adaptive Algorithm that approximates the optimal Non-Adaptive cost within a sublogarithmic factor and we provided a tight upper bound through a simple randomized rounding on the Linear Programming Relaxation of the formulation of Non-Adaptive Algorithms. We also defined a noisy setting of the original problem where algorithms can query an oracle with one-sided error. We proved that lower bounds of the noisy setting are transferred to the original problem and we formulated the noisy setting as a Markov Decision Process. As a result, we were able to compute the optimal policy for small instances and computationally prove a constant lower bound for Fully Adaptive algorithms. Finally, we designed Adaptive algorithms that achieve a constant approximation of the optimal Non-Adaptive cost in graphs of bounded treewidth and also in spare & minor-closed graphs.

It will be interesting to examine whether the $O(k)$ competitive ratio for graphs of treewidth k can be improved to $O(\log k)$, in order to gracefully degrade to the brute force competitive ratio as the treewidth increases, or whether there exists a parameterized lower bound. Apart from that, it will also be interesting to provide a more fine-grained competitive ratio with respect to characteristics of the inputs distribution (e.g. its entropy). The difficulty in this problem is

"two-dimensional", with the two independent dimensions being the moldgraph and the input distribution. It seems that for a fixed moldgraph, the problem becomes harder as the entropy of the input distribution increases. Similarly, if we restrict to a certain entropy level, e.g. to uniform distributions over all spanning trees, the difficulty of the problem seems to increase as the moldgraph's treewidth increases. It will be interesting to try to formally prove these intuitive results thus providing a very clear view on the problem. Last but not least, it will be interesting to exploit the Markov Decision Process formulation of the noisy setting and investigate whether Reinforcement Learning can help us design better Adaptive strategies.

Bibliography

[ACCL06] Nir Ailon, Bernard Chazelle, Seshadhri Comandur, and Ding Liu. Self-improving algorithms. volume 40, pages 261–270, 01 2006.

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[Ami10] Eyal Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

[ASW13] Marek Adamczyk, Maxim Sviridenko, and Justin Ward. Submodular stochastic probing on matroids. *Leibniz International Proceedings in Informatics, LIPIcs*, 25, 10 2013.

[Bal21] Maria-Florina Balcan. *Data-Driven Algorithm Design*, page 626–645. Cambridge University Press, 2021.

[BDD+16] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michał Pilipczuk. A $c^k n$ 5-approximation algorithm for treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016.

[BDD+21] Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. STOC 2021, New York, NY, USA, 2021. Association for Computing Machinery.

[BDR14] Shalev Ben-David and Lev Reyzin. Data stability in clustering: A closer look. *Theoretical Computer Science*, 558:51–61, 2014. Algorithmic Learning Theory.

[BDSV18] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 344–353. PMLR, 10–15 Jul 2018.

[BDW18] Maria-Florina F Balcan, Travis Dick, and Colin White. Data-driven clustering via parameterized lloyd's families. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[BEL57] RICHARD BELLMAN. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[BF20] Mahdi Belbasi and Martin Fürer. An improvement of reed's treewidth approximation. *CoRR*, abs/2010.03105, 2020.

[BF21] Mahdi Belbasi and Martin Fürer. Finding all leftmost separators of size $\leq k$, 2021.

[BGK10] Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA '10*, 2010.

[BHW20] Maria-Florina Balcan, Nika Haghtalab, and Colin White. k-center clustering under perturbation resilience. *ACM Trans. Algorithms*, 16(2), mar 2020.

[BK96] Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *J. Algorithms*, 21(2):358–402, September 1996.

[BL16] Maria Florina Balcan and Yingyu Liang. Clustering under perturbation resilience. *SIAM Journal on Computing*, 45(1):102–155, 2016.

[Blu21] Avrim Blum. *Approximation Stability and Proxy Objectives*, page 120–139. Cambridge University Press, 2021.

[BNVW17] Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In Satyen Kale and Ohad Shamir, editors, *Proceedings of the 2017 Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 213–274. PMLR, 07–10 Jul 2017.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[BSV16] Maria-Florina F Balcan, Tuomas Sandholm, and Ellen Vitercik. Sample complexity of automated mechanism design. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[BSV18] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. A general theory of sample complexity for multi-item profit maximization. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, EC '18, page 173–174, New York, NY, USA, 2018. Association for Computing Machinery.

[BW17] Maria-Florina Balcan and Colin White. Clustering under local stability: Bridging the gap between worst-case and beyond worst-case analysis. *ArXiv*, abs/1705.07157, 2017.

[CFG+02] Moses Charikar, Ronald Fagin, Venkatesan Guruswami, Jon Kleinberg, Prabhakar Raghavan, and Amit Sahai. Query strategies for priced information. *Journal of Computer and System Sciences*, 64(4):785–819, 2002.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[CGT+20] Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. Pandora's box with correlations: Learning and approximation. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1214–1225. IEEE, 2020.

[DK18] Dan DeBlasio and John Kececioglu. *Parameter Advising for Multiple Sequence Alignment*. 01 2018.

[ER59] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[FFX+17] Luoyi Fu, Xinzhe Fu, Zhiying Xu, Qianyang Peng, Xinbing Wang, and Songwu Lu. Determining source–destination connectivity in uncertain networks: Modeling and solutions. *IEEE/ACM Transactions on Networking*, 25(6):3237–3252, 2017.

[FHL08] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM Journal on Computing*, 38(2):629–657, 2008.

[Fin98] Eugene Fink. How to solve it automatically: Selection among problem-solving methods. 04 1998.

[FLS+18] Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, MichaŁ Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3), June 2018.

[FTV15] Fedor V. Fomin, Ioan Todinca, and Yngve Villanger. Large induced subgraphs via triangulations and cmso. *SIAM Journal on Computing*, 44(1):54–87, 2015.

[GGM06] Ashish Goel, Sudipto Guha, and Kamesh Munagala. Asking the right questions: Model-driven optimization using probes. pages 203–212, 01 2006.

[GJSS19] Anupam Gupta, Haotian Jiang, Ziv Scully, and Sahil Singla. *The Markovian Price of Information*, pages 233–246. 04 2019.

[GNS] Anupam Gupta, Viswanath Nagarajan, and Sahil Singla. *Algorithms and Adaptivity Gaps for Stochastic Probing*, pages 1731–1747.

[GR15] Rishi Gupta and Tim Roughgarden. A pac approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46, 11 2015.

[GR17] Rishi Gupta and Tim Roughgarden. A pac approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

[GV04] Michel X. Goemans and Jan Vondrák. Covering minimum spanning trees of random subgraphs. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, page 934–941. Society for Industrial and Applied Mathematics, 2004.

[How60] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[HRG+13] Eric Horvitz, Yongshao Ruan, Carla Gomes, Henry Kautz, Bart Selman, and David Chickering. A bayesian approach to tackling hard computational problems. 01 2013.

[KM70] Victor Klee and George J. Minty. How good is the simplex algorithm? 1970.

[KMMO90] 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*, SODA '90, page 301–309, USA, 1990. Society for Industrial and Applied Mathematics.

[Kor21] Tuukka Korhonen. Single-exponential time 2-approximation algorithm for treewidth. *CoRR*, abs/2104.07463, 2021.

[Lag96] Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *Journal of Algorithms*, 20(1):20–44, 1996.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, September 1994.

[MM21] Konstantin Makarychev and Yury Makarychev. *Perturbation Resilience*, page 95–119. Cambridge University Press, 2021.

[MR15] Jamie Morgenstern and Tim Roughgarden. On the pseudo-dimension of nearly optimal auctions. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 136–144, 2015.

[Ree92] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 221–228, New York, NY, USA, 1992. Association for Computing Machinery.

[Rou21] Tim Roughgarden. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021.

[RS86] Neil Robertson and P.D Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

[RS95] N. Robertson and P.D. Seymour. Graph minors .xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

[San03] Tuomas Sandholm. Automated mechanism design: A new application area for search algorithms. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming – CP 2003*, pages 19–36, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[SS08] Frans Schalekamp and David B. Shmoys. Algorithms for the universal and a priori tsp. *Oper. Res. Lett.*, 36(1):1–3, jan 2008.

[SW11] Martin Skutella and David Williamson. A note on the generalized min-sum set cover problem. *Operations Research Letters - ORL*, 39, 07 2011.

[Val84] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984.

[Vap99] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 1999.

[Wei79] Martin L. Weitzman. Optimal search for the best alternative. *Econometrica*, May, 47(3):641–654, 1979.

[XHHLB08] Lin Xu, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 06 2008.

Appendix A

Tree Decompositions

In this section we introduce the notion of a *Tree Decomposition* and define the *Treewidth* of a graph. Treewidth is a fundamental tool that has been heavily used in many graph algorithms, especially in the field of Parametrized Algorithms. Our main result, Algorithm 10, utilizes a Tree Decomposition of the input graph and achieves a constant competitive ratio on graphs of bounded Treewidth.

A.1 Definition of Treewidth

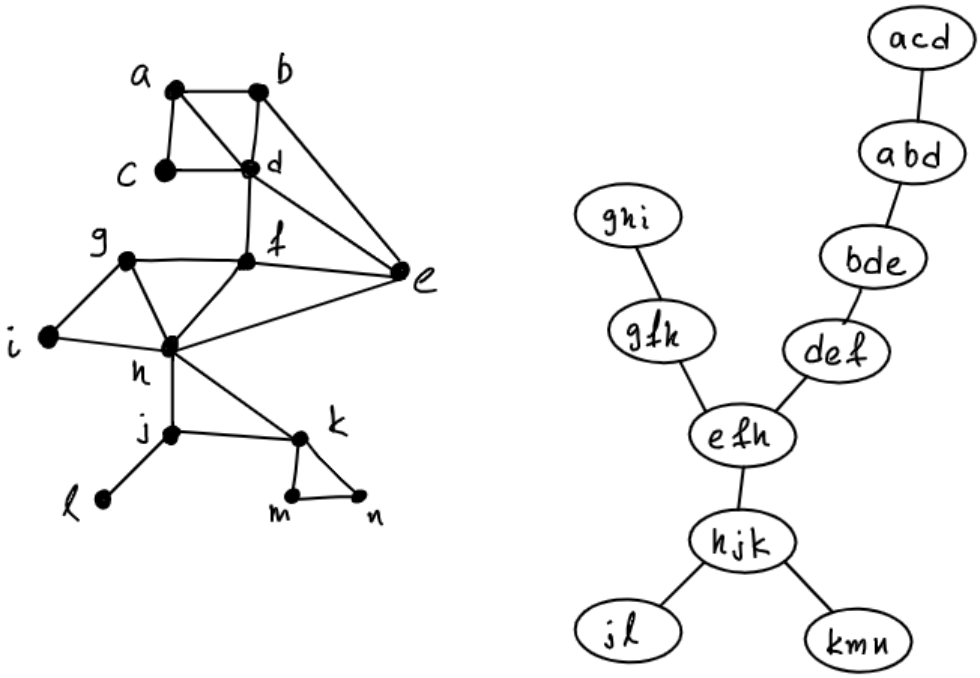
The Tree Decomposition of a Graph, first introduced in [RS86], is a tool that is widely used in the field of Parametrized Algorithms. The motivation for tree decompositions comes from many problems like Vertex Cover or Maximum Independent Set that are NP-Hard in the general case but are easily solvable on trees. Researchers tried to find properties of graphs that "measure" their tree-likeness, that is how "far" a graph G is from being a tree. There are many ways to measure the "tree-likeness" of a graph G. For instance, one could count the number of cycles of G, or the minimum number of edges that should be removed in order for G to become a tree. However, the approach that proved the most useful was the decomposition of G into buckets of vertices that are connected in a tree-like fashion, which is called a tree decomposition of G. The treewidth of a graph G is simply a quantitative measure of how good a tree decomposition we can obtain. In the following lines we will formally define the tree-decomposition of a graph G. A reader interested in an in-depth view of tree decompositions is encouraged to read Chapter 7 of [CFK$^+$15].

Definition A.1.1 (Tree Deecomposition)**.** A *Tree Decomposition* of a graph G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ where T is a tree whose every node t is a assigned a vertex subset $X_t \subseteq V(G)$, called a bag, such that the following three conditions hold:

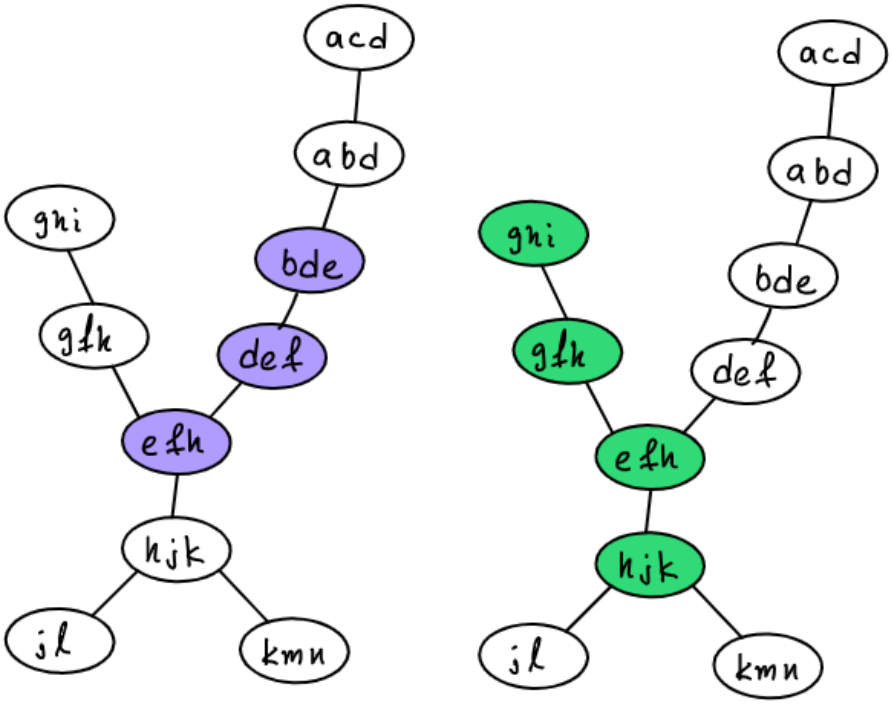
1. $\cup_{t \in V(T)} X_t = V(G)$. In other words, every vertex of G is in at least one bag.
2. For every edge $(u, v) \in E(G)$, there exists a node t of T such that bag X_t contains both u and v.
3. For every $u \in V(G)$, the set $T_u = \{t \in V(T) : u \in X_t\}$, that is the set of nodes whose corresponding bags contain u, induces a connected subtree of T

The width of tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ equals $\max_{t \in V(T)} |X_t| - 1$, that is the maximum bag size minus 1. Of course, we can have tree decompositions of various widths for a given graph. For example, putting all the vertices of a graph G in a single node is a valid tree decomposition of G, however it gives us no additional information about G. Therefore, we are interested in the "best" tree decomposition we can obtain from G. This is captured by the *Treewidth* of G, denoted by tw(G), which is the minimum possible width of a tree decomposition of G.

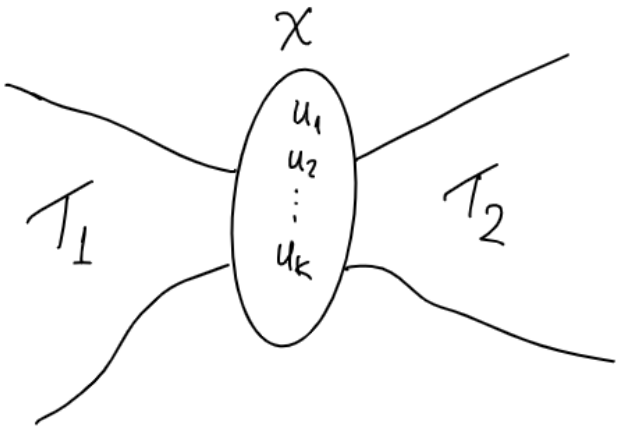
In the following figure, you can see an example of a tree decomposition of a graph. Notice how neighboring vertices are collected in a bag in order to form a tree of bags.



All valid tree decompositions have the property that the for all vertices u of the graph G, the induced subtree of the tree decomposition whose sets include u is connected. This is illustrated in the following figure.



This is a very useful property, especially for connectivity problems. Let t be a node of a tree decomposition T and suppose that t has only two children as depicted below.



The former property guarantees that all the vertices belonging to bags in T_1 are connected to the vertices belonging to bags in T_2 through the vertices X_t in G. It is trivial to prove the latter, since the vertices in $V(T_1) \setminus X_t$ and the vertices in $V(T_2) \setminus X_t$ do not appear in any common bags and therefore there are no edges between them in the graph G. As a result, the vertices in X_t are a separator for the graph G.

A.2 Calculating the Treewidth of a graph

The decision problem of determining whether a given graph G has Treewidth less than a given number k was proven to be NP-Complete in [ACP87]. However, there are many algorithms that are exponential in the parameter k but polynomial in the number of vertices and edges of G that can find a tree decomposition of a graph G. For graphs that have bounded treewidth, these algorithms run in polynomial time.

There are also many algorithms that construct tree decompositions of a given graph with a width that approximates its treewidth. These algorithms typically have better running time guarantees. Depending on the application on hand, one might prefer a better approximation ratio or a better dependence in the running time from the size of the input or the treewidth. The table below provides an overview of some treewidth algorithms. We use k to denote the treewidth of the input graph and n to denote the number of its vertices. Each of these algorithms outputs a tree decomposition of width given in the Approximation column, in running time $f(k) \cdot g(n)$.

Approximation	$f(k)$	$g(n)$	**Reference**
Exact	$O(1)$	$O(n^{k+2})$	[ACP87]
$4k+3$	$O(3^{3k})$	$O(n^2)$	[RS95]
$8k+7$	$2^{O(k \log k)}$	$O(n \log^2 n)$	[Lag96]
$5k+4$	$2^{O(k \log k)}$	$O(n \log n)$	[Ree92]
Exact	$k^{O(k^3)}$	$O(n)$	[Bod96]
$O(k\sqrt{\log k})$	$O(1)$	$n^{O(1)}$	[FHL08]
$4.5k+4$	2^{3k}	$O(n \log^2 n)$	[Ami10]
$\frac{11}{3}k+4$	$2^{3.6982k}$	$O(n^2)$	[Ami10]
Exact	$O(1)$	$O(1.7347^n)$	[FTV15]
$3k+2$	$2^{O(k)}$	$O(n \log n)$	[BDD$^+$16]
$5k+4$	$2^{O(k)}$	$O(n)$	[BDD$^+$16]
k^2	$O(k^7)$	$O(n \log n)$	[FLS$^+$18]
$5k+4$	$2^{8.765k}$	$O(n \log n)$	[BF20]
$2k+1$	$2^{O(k)}$	$O(n)$	[Kor21]
$5k+4$	$2^{6.755k}$	$O(n \log n)$	[BF21]

A.3 "Nice" Tree Decompositions

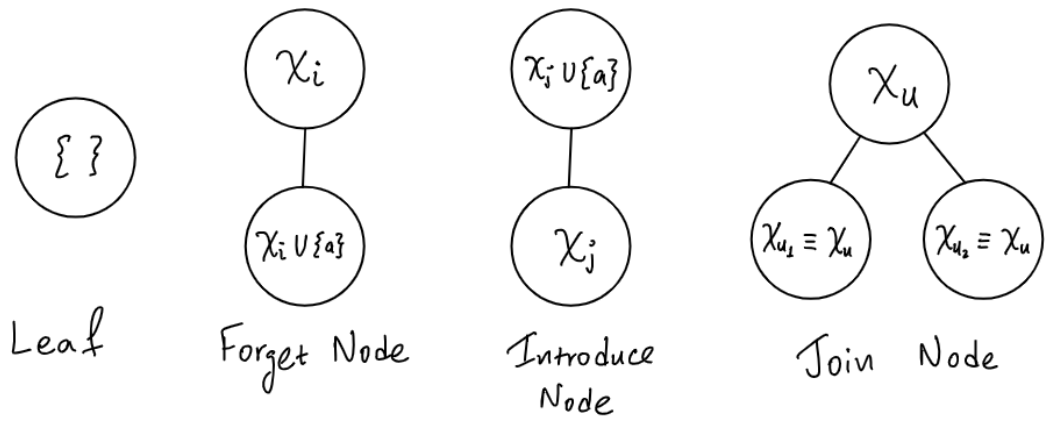
In this section, we present a more restricted version of a tree decomposition that is useful for designing algorithms. These decompositions are called "Nice" Tree Decompositions.

Definition A.3.1 ("Nice" Tree Decomposition)**.** A tree decomposition of a graph G is called a *"Nice" Tree Decomposition* if the following conditions are satisfied:

1. Every node of the tree decomposition has at most two children
2. If a node u has two children i and j then it holds that $X_u = X_i = X_j$
3. If a node i has one child j, then either $|X_i| = |X_j| + 1$ and $X_j \subset X_i$ or $|X_i| = |X_j| - 1$ and $X_i \subset X_j$

Based on Definition A.3.1, every node of a "Nice" Tree Decomposition of a graph G will fall in one of the following categories:

1. **Leaf**: Nodes that contain an empty bag, $X = \emptyset$
2. **Introduce Node**: A node i with exactly one child j, such that $X_i = X_j \cup \{u\}$ for some vertex $u \in V(G)$ and $u \notin X_j$
3. **Forget Node**: A node i with exactly one child j, such that $X_i = X_j \setminus \{u\}$ for some vertex $u \in X_j$
4. **Join Node**: A node u with exactly two children $u_1,\ u_2$ such that $X_u = X_{u_1} = X_{u_2}$



These decompositions might be very useful for algorithm designers, however it is crucial to investigate whether all graphs admit such decompositions, how their width relates to the graph's treewidth and whether we can compute them in a reasonable time. All these questions were answered in [BK96] and are summed up in the following Lemma.

Lemma A.3.2 (Lemma 2.2 of [BK96])**.** *Every graph G with treewidth k has a "Nice" Tree Decomposition of width k that can be obtained in $O(n)$ time.*

Due to Lemma A.3.2, it suffices to design our algorithms on "Nice" Tree Decompositions and it is guaranteed that those will be of optimal width.

Appendix B

Ommited Proofs

Proof of Lemma 4.4.7. We will prove Lemma 4.4.7 by induction on the subtrees of the tree decomposition. It is clear that the proposition holds for Leafs $u \in T$ that contain bags of only one vertex, $|X_u| = 1$. Assume that the Lemma holds for the subtrees rooted at the children of a node $i \in T$. Node i can have only one of the following types:

1. If i is an *Introduce Node*, then it will have exactly one child j with $X_i = X_j \cup \{a\}$ for some vertex $a \in G$ for which $a \notin X_j$. From the induction hypothesis, we can form $|X_j|$ components with the vertices $V(T_j)$ using only instantiated edges from $E(T_j)$. Adding a new component that only contains a, gives us $|X_j| + 1 = |X_i|$ components with all the vertices from $V(T_i) = V(T_j) \cup \{a\}$.

2. If i is a *Join Node*, then it will have exactly two children i_1 and i_2 with $X_i = X_{i_1} = X_{i_2}$. From the induction hypothesis, we can form $|X_{i_1}|$ components with $V(T_{i_1})$ and $|X_{i_2}|$ components with $V(T_{i_2})$. Note that from our invariant, each connected component will contain at least one vertex from the root's bag. Therefore, each of the $|X_{i_1}|$ connected components, will share a vertex with at least one of the $|X_{i_2}|$ components. After merging these components, we will have at most $|X_{i_1}| = |X_i|$ components.

3. If i is a *Forget Node*, then it will have exactly one child j with $X_i = X_j \setminus \{a\}$ for some vertex $a \in G$. From the induction hypothesis, we can form at most $|X_j|$ components using all the vertices in $E(T_j) = E(T_i)$. Let C be the connected component that includes a. We consider the following cases:

   (a) If the connected component C contains another vertex from $|X_j|$, that is $C \cap X_j \setminus \{a\} \neq \emptyset$, then the remaining components will be at most $|X_j| - 2$ since due to our invariant, each one of them will have to contain at least one vertex from $|X_j|$. Therefore, the total number of components is at most $|X_j| - 1 = |X_i|$.

   (b) If $C \cap X_j = \{a\}$, then it holds that $C \cap X_i = \emptyset$. From the properties of the tree decomposition, all the nodes that contain a vertex must form a continuous path. Since $C \cap X_i = \emptyset$ and $C \subset E(T_j)$, all the nodes, from the whole tree decomposition, that contain the vertices of C belong to the subtree T_j. Therefore, all edges that are adjacent to vertices in C are included in $E(T_j)$, that is

$$\{(u, v) \in E(G) : u \in C \text{ or } v \in C\} \subset E(T_j) = E(T_i)$$

This means that $E(T_i)$ contains a cut that separates C from $V(G) \setminus C$. We know that all instantiated graphs are connected, therefore $E(T_i)$ contains at least one instantiated edge that covers this cut and connects C to another of the at most $|X_j| - 1$ remaining components. Therefore, including this instantiated edge, we can form at most $|X_i|$ connected components that include all the vertices of $V(T_i)$ using only instantiated edges from $E(T_i)$.

□