Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Ανάθεση Αντικειμένων και Εξωτερικές Επιδράσεις σε Γράφους

Object Allocations and Graph Externalities

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΤΥΛΙΑΝΟΣ Α. ΚΑΣΟΥΡΙΔΗΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Τομέας Τεχνολογίας Πληροφορικής
και Υπολογιστών

Ανάθεση Αντικειμένων και Εξωτερικές Επιδράσεις σε Γράφους

Object Allocations and Graph Externalities

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΤΥΛΙΑΝΟΣ Α. ΚΑΣΟΥΡΙΔΗΣ

Επιβλέπων : Αριστείδης Παγουρτζής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 9η Ιουλίου 2021.

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

Αθήνα, Ιούλιος 2021

Στυλιανός Α. Κασουρίδης

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

-

Περίληψη

Θεωρούμε την επίδραση ετερογενών αντικειμένων, τοποθετημένων σε διάφορες τοποθεσίες σε ένα δίκτυο, σε γειτονικά αντικείμενα ή οντότητες. Εισάγουμε και μελετάμε ένα μοντέλο, στο οποίο αντικείμενα με υψηλή αξία ασκούν θετική εξωτερική επίδραση σε γειτονικά αντικείμενα με μικρότερη αξία. Στόχος είναι η μεγιστοποίηση αυτής της θετικής επίδρασης, την οποία ονομάζουμε συνολική εξωτερική επίδραση στο γράφημα.

Αρχικά αποδεικνύουμε ότι το πρόβλημα είναι NP-hard, όταν ο μέγιστος βαθμός είναι 3, αξιοποιώντας τη σχέση με το πρόβλημα του ελάχιστου κυρίαρχου συνόλο (minimum dominating set). Στη συνέχεια, δείχνουμε ότι το πρόβλημα είναι επιλύσιμο σε πολυωνυμικό χρόνο όταν ο μέγιστος βαθμός είναι 2. Επιπλέον παρουσιάζουμε αποδοτικούς ακριβείς και προσεγγιστικούς αλγορίθμους για διάφορες ειδικές περιπτώσεις και τοπολογίες. Συγκεκριμένα, δείχνουμε ότι αν υπάρχουν μόνο δύο διαφορετικές πιθανές αξίες αντικειμένων, τότε μια φυσιολογική άπληστη στρατηγική, η οποία πετυχαίνει καλά αποτελέσματα για προβλήματα μέγιστης κάλυψης, οδηγεί σε ένα προσεγγιστικό αλγόριθμο με σταθερό λόγο προσέγγισης.

Διεξάγουμε εκτεταμένα αριθμητικά πειράματα, μέσω των οποίων δείχνουμε ότι ο άπληστος αλγόριθμος πετυχαίνει πολύ καλά αποτελέσματα για τη γενική περίπτωση του προβλήματος. Θεωρούμε περαιτέρω γενικεύσεις του προβλήματος σε άλλα μοντέλα εξωτερικής επίδρασης και αξιολογούμε την άπληστη προσέγγιση σε αυτές τις πιο γενικές περιπτώσεις.

Λέξεις κλειδιά

αλγοριθμική θεωρία, θεωρία γραφημάτων, εξωτερικές επιδράσεις, καλύψεις, κυρίαρχο σύνολο, υπολογιστική πολυπλοκότητα, προηγμένα θέματα αλγορίθμων, προσεγγιστικοί αλγόριθμοι, πειραματική αξιολόγηση αλγορίθμων

Abstract

We consider the influence of heterogeneous objects placed in various locations in a network to neighboring objects or entities. We introduce and study a model where elements with high value exert a positive externality on neighboring elements whose value is lower. We aim at maximizing this positive influence called graph externality.

We first prove the NP-hardness of the problem, when the maximum degree is 3, by exploiting a connection to the minimum dominating set problem. We then proceed to show that the problem is polynomial time solvable when the maximum degree is 2. We also present efficient exact and approximation algorithms for several special cases and topologies. In particular, we show that if only two valuations exist, then a natural greedy strategy, which works well for maximum coverage problems, leads to a constant approximation algorithm.

We conduct extensive numerical experiments that show that the greedy algorithm performs very well for general valuations. We further consider generalizations of the problem to other externality models and evaluate the greedy approach under these more general settings.

Key words

algorithm theory, graph theory, externalities, coverings, dominating set, computational complexity, advanced algorithms, approximation algorithms, experimental evaluation of algorithms

Ευχαριστίες

Αρχικά, θέλω να ευχαριστήσω από καρδιάς τον επιβλέποντα καθηγητή της διπλωματικής μου εργασίας, κ. Αριστείδη Παγουρτζή, τόσο για την επίβλεψη, όσο και για τη συνεχή βοήθεια, τις συνεχείς συμβουλές και την καθοδήγηση που μου παρείχε, σε όποιο σημείο χρειαζόταν, με εύστοχες παρεμβάσεις και διαθέτοντας αρκετό χρόνο, ακόμα και ώρες, σε περίπτωση που οι συνθήκες το απαιτούσαν. Ένας εξαιρετικός καθηγητής, ο οποίος στάθηκε δίπλα μου και με στήριξε σε δύσκολες καταστάσεις, τόσο για την εργασία όσο και με τις επιλογές για τις σπουδές μου, βοηθώντας με πάντα να συνεχίσω το έργο μου, να συνεχίζω με δύναμη και να παίρνω τις καλύτερες αποφάσεις με γνώμονα πάντα το καλό μου. Του είμαι πραγματικά ευγνώμων και ελπίζω και εύχομαι η εξαιρετική συνεργασία μας να συνεχιστεί και στο μέλλον.

Επιπλέον, θέλω να ευχαριστήσω θερμά τον κ. Δημήτριο Φωτάκη για την επίσης πολύτιμη βοήθεια και συμβολή του, τόσο για την παρούσα εργασία, όσο και για τις χρήσιμες ιδέες, συμβουλές και παρατηρήσεις του. Ο κ. Φωτάκης ήταν και είναι ένας άνθρωπος που θα μπορούσα ανά πάσα στιγμή να συμβουλευτώ για οτιδήποτε χρειαζόμουν, γνωρίζοντας εκ των προτέρων ότι θα μου μιλήσει με -αφοπλιστική- ειλικρίνεια και σοβαρότητα, ακόμα και για τα πιο απλά θέματα. Ένας εκπληκτικός καθηγητής και επιστήμονας, που πάντοτε άκουγα ό,τι μου έλεγε με πλήρη προσοχή, με τον οποίο θέλω και ελπίζω να έχουμε και μελλοντικές συνεργασίες.

Στο σημείο αυτό, θέλω να απονείμω ιδιαίτερες ευχαριστίες στον κ. Αντώνιο Συμβώνη, ένας καθηγητής ο οποίος έχει εξαιρετική μεταδοτικότητα στη διδασκαλία του, όντας ευγνώμων που ήμουν μαθητής του στο μάθημα της Θεωρίας Γραφημάτων, που έχει άμεση σχέση με την παρούσα διπλωματική εργασία. Ένας ευγενέστατος άνθρωπος, ο οποίος πάντοτε εξηγούσε τα πάντα λεπτομερώς και με σαφήνεια.

Θέλω να ευχαριστήσω πολύ και τον κ. Laurent Gourvès, του πανεπιστημίου Université Paris-Dauphine, PSL. Είναι τιμή μου που συνεργάστηκα μαζί με τον κ. Gourvès, τον κ. Παγουρτζή και τον κ. Φωτάκη για τη δημοσίευση στη βάση της οποίας εκπονήθηκε η παρούσα διπλωματική εργασία έγινε δεκτή στο συνέδριο European Conference of Artificial Intelligence 2020.

Θέλω ακόμη να ευχαριστήσω όλους τους καθηγητές με τους οποίους συναναστράφηκα και μου μετέδωσαν αρκετά πράγματα καθ' όλη τη διάρκεια των σπουδών μου. Η συμβολή τους είναι ιδιαίτερα πολύτιμη και χαίρομαι πραγματικά που είχα την τιμή να υπάρξω φοιτητής από τόσο αξιόλογους επιστήμονες.

Η παρούσα εργασία σημαίνει το τέλος ενός πολύ σημαντικού κύκλου της ζωής μου, της φοίτησής μου στο Εθνικό Μετσόβιο Πολυτεχνείο. Οι συναναστροφές μου και οι φίλοι που έκανα σε αυτό τον κύκλο μου είναι ιδιαίτερα πολύτιμες, καθώς θεωρώ ότι έχω γνωρίσει ανθρώπους που

είναι ξεχωριστοί, οι οποίοι ήταν δίπλα μου και σε δύσκολες στιγμές και σε εύκολες. Είμαι ιδιαίτερα ευχαριστημένος που έχω διατηρήσει επαφές με τους ανθρώπους αυτούς παρά τις ιδιαίτερες συνθήκες της πανδημίας, και λόγω αυτού, σε συνδυασμό με τα πράγματα που μας ενώνουν, τον ενθουσιασμό στις ομορφες στιγμές που έχουμε περάσει, και τη συμπαράσταση στις δύσκολες στιγμές, πιστεύω ότι είναι αρκετά πιθανό να διατηρηθούν οι επαφές αυτές παρά το γεγονός ότι δε θα συναντιόμαστε στο πανεπιστήμιο. Γνωρίζουν όλοι αυτοί ποιοι είναι και θέλω να ευχαριστήσω τον καθένα τους ξεχωριστά.

Πέρα από τις γνωριμίες που έκανα στο πανεπιστήμιο, θέλω να ευχαριστήσω θερμά τους φίλους μου που γνώριζα πριν το πανεπιστήμιο, με τους οποίους ακόμα κάνουμε πολύ ισχυρή παρέα και πιστεύω ότι θα συνεχίσουμε να κάνουμε για αρκετό καιρό. Ήταν πάντα δίπλα μου όταν τους χρειαζόμουν και όσα χρόνια κι αν περάσουν, γνωρίζουμε ότι αρκετές συζητήσεις μας και ο χρόνος που περνάμε μαζί θα συνεχίσουν να είναι όπως όταν ήμασταν παιδιά. Αυτό μας δίνει δύναμη τόσο σε εμένα, όσο και σε αυτούς, αλλά και αμέτρητη χαρά.

Τέλος, θέλω να ήθελα να ευχαριστήσω ιδιαίτερα την οικογένεια μου, πρώτα τους γονείς μου, οι οποίοι πέρα από το γεγονός ότι πάντα, ακόμα και από πολύ μικρή ηλικία, με στήριζαν σε όλες τις αποφάσεις μου, φρόντισαν να με συμβουλεύουν πάντα για το καλύτερο και να με καθοδηγούν με το σωστότερο τρόπο όπου χρειαζόταν ότι απαιτείται. Θεωρώ ότι χωρίς αυτούς, δε θα ήμουν ο άνθρωπος που είμαι σήμερα, και με έχουν βοηθήσει ιδιαίτερα τόσο στην ακαδημαϊκή/σχολική μου πορεία, όσο και στην πορεία μου ως χαρακτήρας. Θέλω να ευχαριστήσω και την αδερφή μου, η οποία πάντα μου δίνει χαρά και σημαντική στήριξη, κι ας είναι αρκετά χρόνια μικρότερη, και να της ευχηθώ καλή επιτυχία, καθώς του χρόνου θα δώσει και αυτή το δικό της αγώνα στις Πανελλήνιες! Ευχαριστώ ακόμα όλα τα άλλα μέλη της οικογένειάς μου, τον παππού μου, τη γιαγιά μου, τους θείους μου, που πάντα ήταν και είναι δίπλα μου, νοιάζονται πραγματικά για εμένα και πάντα χαίρομαι να τους βλέπω και να περνάω χρόνο μαζί τους. Νιώθω βαθιά συγκινημένος για τη στήριξη και την αγάπη που έχω λάβει απο την οικογένειά μου και ελπίζω να τους κάνω υπερήφανους.

Στυλιανός Α. Κασουρίδης,

Αθήνα, 9η Ιουλίου 2021

Περιεχόμενα

Κατάλογος σχημάτων

Κεφάλαιο 1

Εκτεταμένη Περίληψη στα Ελληνικά

Στο κεφάλαιο αυτό, περιγράφονται πιο συνοπτικά από τα επόμενα κεφάλαια οι ερευνητικές έννοιες της εργασίας. Έχει γίνει προσπάθεια το περιεχόμενο αυτού του κεφαλαίου να είναι όσο το δυνατόν επεξηγηματικό και να δίνει βάση σε όλες τις έννοιες που αναλύονται στην εργασία, αλλά σε περίπτωση που μεγαλύτερη ανάλυση κριθεί απαραίτητη από τον αναγνώστη, συνιστάται αναδρομή στην ανάλυση των επόμενων κεφαλαίων.

1.1 Βασικές Έννοιες

1.1.1 Θεωρία Γραφημάτων

Ορίζουμε ως ένα **γράφο** $G = (V, E)$ ως μια δομή δεδομένων η οποία μοντελοποιεί τις σχέσεις μεταξύ συγκεκριμένων ζευγαριών αντικειμένων. Συγκεκριμένα, το σύνολο V είναι το σύνολο των **κορυφών** ή **κόμβων**, ενώ το σύνολο E είναι το σύνολο των **ακμών**.

Ένα **απλό** γράφημα ορίζεται ως ένα γράφημα στο οποίο δεν υπάρχουν δύο διαφορετικές ακμές που συνδέουν τις ίδιες κορυφές. Θα θεωρήσουμε ότι όλα τα γραφήματα στην παρούσα εργασία είναι απλά.

Αν οι ακμές ενός γραφήματος δεν έχουν συγκεκριμένη κατεύθυνση, το γράφημα ορίζεται ως **μη κατευθυνόμενο**. Αν υπάρχουν κατευθυνόμενες ακμές (τις οποίες ονομάζουμε **τόξα**), το γράφημα θεωρείται **κατευθυνόμενο**. Μία ακμή που συνδέει τις κορυφές v_1, v_2 γράφεται με αγκύλες, $\{v_1, v_2\}$, ενώ ένα τόξο από την κορυφή v_1 στην κορυφή v_2 γράφεται με παρενθέσεις (v_1, v_2). Αν δύο κορυφές σε ένα μη κατευθυνόμενο γράφημα v_1, v_2 συνδέονται με ακμή, λέμε ότι η μία κορυφή είναι **γείτονας** της άλλης. Το σύνολο των γειτόνων μιας κορυφής v ορίζεται ως **γειτονιά** της v και συμβολίζεται με $N(v)$, ενώ η **κλειστή γειτονιά** της v είναι η ένωση της v με το σύνολο $N(v)$ και συμβολίζεται με $N[v]$.

Ο **βαθμός** μιας κορυφής είναι το πλήθος των γειτόνων της. Σε κατευθυνόμενα γραφήματα, ορίζουμε τον **έσω βαθμό** ως το πλήθος των τόξων που καταλήγουν στη v και τον **έξω βαθμό** ως το πλήθος των τόξων που ξεκινούν από τη v.

Λέμε ότι υπάρχει ένα **μονοπάτι** μεταξύ 2 κορυφών v_1, v_2, αν μέσω διαδοχικών ακμών ή τόξων μπορούμε να φτάσουμε από τη v_1 στη v_2. Ένα γράφημα στο οποίο οποιεσδήποτε 2 κορυφές

συνδέονται με μονοπάτι ονομάζεται **συνεκτικό**, ενώ στην αντίθετη περίπτωση (στην οποία υπάρχουν 2 τουλάχιστον κορυφές που δε συνδέονται με μονοπάτι) το γράφημα ονομάζεται **μη συνεκτικό**.

Ως **υπογράφημα** ενός γραφήματος G_1 ορίζεται ένα γράφημα G_2 που προκύπτει διαγράφοντας ορισμένες κορυφές ή και ακμές του G_1 (ή και καμία). Στην περίπτωση που έχουν διαγραφεί μόνο κορυφές (και οι προσπίπτουσες σε αυτές ακμές) το G_2 είναι **επαγόμενο υπογράφημα**, ενώ στην περίπτωση που έχουν διαγραφεί μόνο ακμές (και οι κορυφές είναι οι ίδιες) το G_2 είναι **παραγόμενο υπογράφημα**. Ως **σύνθλιψη** μιας ακμής $\{v_1, v_2\}$ σε ένα γράφημα G ορίζουμε τη διαγραφή της ακμής και των κορυφών v_1, v_2, οι οποίες αντικαθίστανται από μια νέα κορυφή v που έχει ως ακμές όλες τις ακμές που προσέπιπταν στις v_1, v_2 εκτός από την ακμή $\{v_1, v_2\}$. Ως **ελάσσον** γράφημα ενός γραφήματος G_1 ορίζεται ένα γράφημα G_2 που είναι υπογράφημα του G_1 και ενδεχομένως περιλαμβάνει και συνθλίψεις ακμών από το G_1.

Ως **κύκλο** εννοούμε ένα μονοπάτι το οποίο αρχίζει και περατώνεται στην ίδια κορυφή, χωρίς να περιέχει την ίδια ακμή ή κάποια άλλη κορυφή 2 φορές. Ένα γράφημα χωρίς κύκλους ονομάζεται **δέντρο**. Ορίζουμε μια ακμή ως **χορδή** ενός κύκλου C, αν η ακμή συνδέει δύο κορυφές που είναι μέλη του C αλλά όχι διαδοχικές σε αυτόν. Ένα γράφημα ονομάζεται **χορδικό** αν όλοι οι κύκλοι μεγέθους τουλάχιστον 4 περιέχουν χορδή. Το σύνολο ακμών που προστίθεται για να γίνει ένα γράφημα χορδικό ονομάζεται **χορδική συμπλήρωση** του γραφήματος. Μία χορδική συμπλήρωση σε ένα γράφημα ενδέχεται να μην είναι η μοναδική.

Ως **κλίκα** ονομάζουμε ένα γράφημα ή υποσύνολο κορυφών στο οποίο ανά 2 οι κορυφές συνδέονται με ακμή. Η **μέγιστη κλίκα** ενός γραφήματος είναι το μέγιστο πλήθος κορυφών που σχηματίζουν κλίκα.

Σε ένα γράφημα, αν θεωρήσουμε τη χορδική συμπλήρωση που ελαχιστοποιεί το μέγεθος της μέγιστης κλίκας (ενδεχομένως δεν είναι μοναδική) ως S, τότε η τιμή $|S| - 1$ ορίζεται ως **πάχος δέντρου** (στα αγγλικά **treewidth**), όρος στον οποίο υπάρχει εκτενής βιβλιογραφική μελέτη.

Ένα γράφημα ονομάζεται **επίπεδο** αν γίνεται να σχεδιαστεί με τρόπο ώστε οι ακμές να τέμνονται μόνο στις κορυφές. Ένα γράφημα ονομάζεται **ελαχιστικά μη επίπεδο** αν υπάρχει κορυφή, που με την αφαίρεση της, το επαγόμενο υπογράφημα είναι επίπεδο. Ένα γράφημα G_1 ονομάζεται **απαγορευμένο**, αν με την ύπαρξη του G_1 ως ελάσσον γράφημα του G_2, το G_2 δε μπορεί να έχει ορισμένες ιδιότητες. Για παράδειγμα, ένας κύκλος είναι *απαγορευμένο* ελάσσον γράφημα για τα δέντρα.

Τέλος, ορίζουμε ως **ταίριασμα** σε ένα γράφημα $G = (V, E)$, ένα σύνολο $M \subseteq E$ ώστε οι ακμές του M να μην περιέχουν κοινές κορυφές. Ένα ταίριασμα μέγιστης πληθικότητας σε ένα γράφημα ονομάζεται **μέγιστο ταίριασμα**, ενώ ένα ταίριασμα M στο οποίο αν προστεθεί οποιαδήποτε ακμή που δεν ανήκει στο M παύει να είναι ταίριασμα ονομάζεται **μεγιστοτικό ταίριασμα**.

1.1.2 Αλγόριθμοι και Πολυπλοκότητα

Ορίζουμε ως **αλγόριθμο** μια πεπερασμένη διαδικασία καλά ορισμένων εντολών υλοποιήσιμες σε υπολογιστή, με σκοπό να λυθεί μια κλάση προβλημάτων ή να εκτελεστεί ένας υπολογισμός.

Η ταχύτητα των αλγορίθμων μελετάται με όρους **υπολογιστικής πολυπλοκότητας**. Αν θεωρήσουμε ως n το μέγεθος των αντικειμένων της εισόδου, τότε ένας αλγόριθμος με πλήθος βημάτων ανάλογο του n θεωρείται αλγόριθμος **γραμμικής πολυπλοκότητας**, ένας αλγόριθμος με πλήθος βημάτων ανάλογο του n^2 θεωρείται αλγόριθμος **τετραγωνικής πολυπλοκότητας**, ένας αλγόριθμος με πλήθος βημάτων ανάλογο του log_n θεωρείται αλγόριθμος **λογαριθμικής πολυπλοκότητας** κ.ο.κ..

Γενικότερα, ένας αλγόριθμος με πλήθος βημάτων με μεγιστοβάθμιο όρο το πολύ ανάλογο του n^k θεωρείται αλγόριθμος **πολυωνυμικής πολυπλοκότητας**. Ακόμα και οι αλγόριθμοι λογαριθμικής πολυπλοκότητας θεωρούνται αλγόριθμοι πολυωνυμικής πολυπλοκότητας, επειδή ο χρόνος εκτέλεσής τους φράσσεται ασυμπτωτικά από χρόνους πολυωνυμικής πολυπλοκότητας. Αντίθετα, αλγόριθμοι πλήθους βημάτων ανάλογων του 2^n θεωρούνται αλγόριθμοι **εκθετικής πολυπλοκότητας**. Λέμε ότι ένα πρόβλημα ανήκει στην κλάση **P** αν υπάρχει αλγόριθμος πολυωνυμικής πολυπλοκότητας που το επιλύει ορθά. Η κλάση προβλημάτων για τα οποία δεν έχει βρεθεί πολυωνυμικός αλγόριθμος ακόμα, αλλά μπορεί η ορθότητά μιας λύσης να επαληθευτεί σε πολυωνυμικό χρόνο, ονομάζονται **NP**-πλήρη, ενώ τα προβλήματα για τα οποία δεν έχει βρεθεί πολυωνυμικός αλγόριθμος ακόμα και ούτε η ορθότητά μιας λύσης να επαληθευτεί σε πολυωνυμικό χρόνο, ονομάζονται **NP**-δύσκολα.

1.2 Προηγούμενη Έρευνα

Έχει γίνει έρευνα με προβλήματα σχετικά με αυτά που θα μελετήσουμε, τόσο στον τομέα των οικονομικών όσο και στον τομέα της επιστήμης υπολογιστών.

1.3 Το μοντέλο της εξωτερικής επίδρασης σε γραφήματα

1.3.1 Κίνητρο

Ας σκεφτούμε ένα διευθυντή προγράμματος ενός τηλεοπτικού καναλιού, ο οποίος πρέπει να τοποθετήσει 3 τηλεοπτικά προγράμματα, έστω A, B, C, σε κατάλληλες χρονικές στιγμές, ώστε να μεγιστοποιηθεί η συνολική τηλεθέαση και στα 3 προγράμματα. Ας θεωρήσουμε ότι τα προγράμματα A, B είναι δημοφιλή, ενώ το πρόγραμμα C δεν είναι. Είναι πιθανό επομένως ο διευθυντής να θελήσει να βάλει το πρόγραμμα C μεταξύ των A, B χρονικά, ώστε να πάρει «εξωτερική» επίδραση.

Με παρόμοιο τρόπο, μπορούμε να σκεφτούμε πώς ένας ιδιοκτήτης μιας ιστοσελίδας μπορεί να τοποθετήσει τις ειδήσεις στη σελίδα. Αν η ιστοσελίδα θέλει να προωθεί συγκεκριμένα άρθρα να διαβαστούν, μπορεί να τα βάλει τοπολογικά πιο κοντά σε δημοφιλέστερα άρθρα. Παρόμοια, είθισται οι ακριβότερες διαφημίσεις της σελίδας να είναι αυτές που είναι κοντά σε δημοφιλή άρθρα (τα οποία εμφανίζονται συνήθως με το άνοιγμα της σελίδας), καθώς αντλούν «εξωτερικό» όφελος από αυτά.

1.3.2 Ορισμοί και περιγραφή του μοντέλου

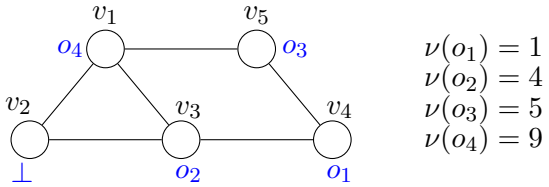
Αν θεωρήσουμε ένα σύνολο m αντικειμένων (μπορούμε να τα σκεφτούμε και ως «αγαθά»), ορισμένο ως $O = \{o_1, o_2, ..., o_m\}$. Κάθε αντικείμενο $o_i \in O$ έχει μια **αξία** $\nu(o_i)$. Θεωρούμε ακόμα ένα γράφημα $G = (V, E)$ με $|V| = n$, στις κορυφές του οποίου θα ανατεθούν τα αντικείμενα του O. Επειδή οι κορυφές θα πρέπει να είναι αρκετές για να μπορούν να ανατεθούν όλα τα αντικείμενα σε αυτές, θεωρούμε ότι ισχύει $n = |V| \leq |O| = m$.

Θεωρούμε ότι σε κάθε κορυφή μπορεί να ανατεθεί το πολύ 1 αντικείμενο. Ορίζουμε μια **ανάθεση** ως μια συνάρτηση $\pi : V \to O \cup \{\perp\}$, στην οποία κάθε αντικείμενο έχει ακριβώς ένα πρόγονο. Με $\pi(v) = \{\perp\}$ ορίζουμε ότι η κορυφή v δεν έχει κανένα αντικείμενο ανατεθειμένο σε αυτή. Η **αντίστροφη συνάρτηση ανάθεσης** π^-1 είναι μια συνάρτηση $\pi^-1 : O \to V$ που ορίζει σε ποια κορυφή έχει ανατεθεί ένα αντικείμενο.

Ορίζουμε ότι μια κορυφή v **λαμβάνει (εξωτερική) επίδραση** από μια κορυφή w υπό μια ανάθεση π σε περίπτωση που και στις 2 κορυφές έχουν ανατεθεί αντικείμενα (έστω $\pi(v_1) = o_1$ και $\pi(v_2) = o_2$), υπάρχει ακμή $\{v, w\}$, ισχύει $\nu(o_1) < \nu(o_2)$ και ισχύει ότι η w έχει τη μεγαλύτερη αξία μεταξύ των γειτόνων της v. Αν δεν ισχύει κάτι από τα παραπάνω, η v δε λαμβάνει επίδραση από τη w. Λόγω των παραπάνω συνθηκών, παρατηρούμε ότι κάθε κορυφή μπορεί να λαμβάνει επίδραση από το πολύ μία κορυφή. Η επίδραση που λαμβάνει η κορυφή v υπό την ανάθεση π συμβολίζεται με $ext_\pi(v)$.

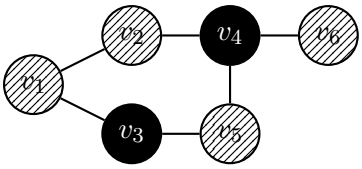
Στην περίπτωση που η v λαμβάνει επίδραση από κάποια κορυφή w, η επίδραση αυτή που λαμβάνει είναι ίση με $\nu(\pi(w)) - \nu(\pi(v))$. Διαφορετικά, η επίδραση που λαμβάνει η v είναι 0. Ορίζουμε την **συνολική επίδραση στο γράφημα** ως το άθροισμα των επιμέρους επιδράσεων που λαμβάνουν όλες οι κορυφές του γραφήματος. Η συνολική επίδραση σε ένα γράφημα $G = (V, E)$ υπό μια ανάθεση π επομένως ισούται με $\sum_{v \in V} ext_\pi(v)$ και συμβολίζεται με $\mathsf{Ext}_\pi(G)$.

Για την περαιτέρω κατανόηση των παραπάνω εννοιών, ας θεωρήσουμε το παρακάτω παράδειγμα με 4 αντικείμενα και τις αντίστοιχες τιμές τους:



$\nu(o_1) = 1$
$\nu(o_2) = 4$
$\nu(o_3) = 5$
$\nu(o_4) = 9$

Η ανάθεση στην παραπάνω εικόνα (κατ' αντιστοιχία με τις κορυφές του γραφήματος) είναι η $\pi = \{o_4, \perp, o_2, o_1, o_3\}$. Η κορυφή v_1 έχει ανατεθειμένο το αντικείμενο o_4 αξίας 9. Δεν υπάρχει κορυφή στη γειτονιά της με μεγαλύτερη αξία, άρα λαμβάνει επίδραση 0. Η κορυφή v_2, επειδή δεν έχει ανατεθειμένο αντικείμενο, λαμβάνει επίσης επίδραση 0. Για τις κορυφές v_3 και v_5, η κορυφή με το αντικείμενο μεγαλύτερης αξίας στη γειτονιά είναι η v_1, άρα λαμβάνουν αξία $\nu(\pi(v_1)) - \nu(\pi(v_3)) = 5$ και $\nu(\pi(v_1)) - \nu(\pi(v_5)) = 4$ αντίστοιχα. Τέλος, με παρόμοιο τρόπο παρατηρούμε ότι η κορυφή v_4 λαμβάνει από την κορυφή v_5 επίδραση $5 - 1 = 4$. Συνολικά, επομένως, η επίδραση στο γράφημα είναι $0 + 0 + 5 + 4 + 4 = 13$.



Σχήμα 1.1: Οι κορυφές v_4 και v_6 συνιστούν κυρίαρχο σύνολο

Μπορούμε να ορίσουμε ότι **διέρχεται επίδραση από μια ακμή** $\{v, w\}$ σε περίπτωση που η κορυφή v λαμβάνει επίδραση από την κορυφή w ή σε περίπτωση που η κορυφή w λαμβάνει επίδραση από τη v (το πολύ ένα από τα δύο μπορεί να ισχύει).

Ορίζουμε επομένως το πρόβλημα OPT-EXT ως εξής: Δοθέντος ενός γραφήματος $G = (V, E)$, ένα σύνολο αντικειμένων O και τις αξίες τους ν, να βρεθεί η ανάθεση π που μεγιστοποιεί το $\mathsf{Ext}_\pi(G)$.

Ένα *στιγμιότυπο* του OPT-EXT ορίζεται ως (G, O, ν), όπου G είναι το γράφημα, O το σύνολο αντικειμένων και ν οι αντίστοιχες αξίες των αντικειμένων.

Μια ανάθεση π^* είναι **βέλτιστη**, σε περίπτωση που η συνολική επίδραση στο γράφημα είναι μέγιστη υπό την ανάθεση π^*. Στην περίπτωση αυτή, η συνολική επίδραση στο γράφημα ονομάζεται **βέλτιστη λύση** για το συγκεκριμένο στιγμιότυπο. Σημείωνεται ότι η βέλτιστη ανάθεση μπορεί να μην είναι μοναδική.

1.3.3 OPT-EXT και πολυπλοκότητα

Ορίζουμε ως **κυρίαρχο σύνολο** σε ένα γράφημα $G = (V, E)$, ένα σύνολο στο οποίο κάθε κορυφή του G είτε ανήκει στη σύνολο, είτε έχει ένα γείτονα στο σύνολο.

Στο παρακάτω γράφημα, παρατηρούμε ότι το σύνολο $\{v_4, v_6\}$ είναι κυρίαρχο σύνολο, καθώς όλες οι κορυφές του γραφήματος που δεν είναι στο σύνολο, είναι γείτονες τουλάχιστον μίας εκ των $\{v_4, v_6\}$.

Παρά το γεγονός ότι το να βρεθεί οποιοδήποτε κυρίαρχο σύνολο είναι τετριμμένο πρόβλημα (ακόμα και το σύνολο V είναι κυρίαρχο), έχει δειχθεί ότι το να βρεθεί κυρίαρχο σύνολο πληθικότητας το πολύ k είναι ένα **NP**-πλήρες πρόβλημα, ακόμα και στην περίπτωση που ο μέγιστος βαθμός του γραφήματος είναι 3, καθώς και στις περιπτώσεις των διμερών (bipartite) και των διαχωρίσιμων (split) γραφημάτων. Το πρόβλημα στο οποίο σε γράφημα G αναζητείται κυρίαρχο σύνολο πληθικότητας το πολύ k ορίζεται ως πρόβλημα DOMINATING SET(G, k).

Υπάρχει σύνδεση του προβλήματος OPT-EXT με το πρόβλημα DOMINATING SET. Συγκεκριμένα, σε περίπτωση που έχουμε ένα στιγμιότυπο (G, O, ν) του προβλήματος OPT-EXT και έναν αριθμό k, το να βρεθεί αν υπάρχει ανάθεση π ώστε να ισχύει $\mathsf{Ext}_\pi(G) \geq k$ είναι **NP**-πλήρες. Αυτό προκύπτει ανάγοντας το πρόβλημα του DOMINATING SET σε πρόβλημα OPT-EXT, με βάση τις παρακάτω δύο μετατροπές:

- Σε περίπτωση που έχουμε ένα στιγμιότυπο του προβλήματος DOMINATING SET(G, k) ,

ας θεωρήσουμε ότι έχουμε k αντικείμενα αξίας 1 και $|V| - k$ αντικείμενα αξίας 0 και ένα κυρίαρχο σύνολο D μεγέθους k. Παρατηρούμε ότι αν κάθε κορυφή από τις k που ανήκουν στο κυρίαρχο σύνολο έχει ένα αντικείμενο αξίας 1, τότε όλες οι άλλες κορυφές έχουν ένα αντικείμενο αξίας 0. Όμως, κάθε μία από τις υπόλοιπες κορυφές έχει ως γείτονα μια κορυφή στο D, που σημαίνει ότι η επίδραση που λαμβάνει θα είναι $1 - 0 = 1$. Άρα, η συνολική επίδραση στο γράφημα θα είναι $\mathsf{Ext}_\pi(G) \geq |V| - k$.

- Αντίστροφα, αν έχουμε μια ανάθεση π τέτοια ώστε $\mathsf{Ext}_p i(G) \geq |V| - k$, θεωρούμε ως D το σύνολο των κορυφών με $\nu(\pi(v)) = 1$. Αφού γνωρίζουμε ότι ήδη k αντικείμενα έχουν αξία 1, ότι κάθε αντικείμενο που έχει αξία 0 δε μπορεί να λάβει επίδραση πάνω από 1 και ότι η συνολική επίδραση στο γράφημα είναι τουλάχιστον $|V| - k$, πρέπει όλα τα αντικείμενα αξίας 0 να λαμβάνουν επίδραση 1. Άρα το σύνολο D είναι κυρίαρχο σύνολο στο G.

Λόγω της δυσκολίας του προβλήματος OPT-EXT, χρησιμοποιούμε προσεγγιστικούς αλγορίθμους και συγκεκριμένες παραλλαγές για τη μελέτη του. Μπορούμε να ποσοτικοποιήσουμε την ποιότητα μιας ανάθεσης (και άρα μιας λύσης) για το OPT-EXT, συγκρίνοντας το με τη βέλτιστη λύση $\mathsf{Ext}_{\pi^*}(G)$ που προκύπτει από τη βέλτιστη ανάθεση π^*, η οποία είναι **NP**-πλήρες να υπολογιστεί. Ορίζουμε μια ανάθεση π ως **ρ-προσεγγιστική** αν ισχύει ότι $\mathsf{Ext}_\pi(G) \geq \rho\mathsf{Ext}_{\pi^*}(G)$. Ένας αλγόριθμος που επιστρέφει μια ρ-προσεγγιστική λύση σε πολυωνυμικό χρόνο ονομάζεται **ρ-προσεγγιστικός**.

1.4 Το πρόβλημα OPT-EXT με δύο αξίες

1.4.1 Περιγραφή και κίνητρο

Ας θεωρήσουμε την παραλλαγή του OPT-EXT στην οποία τα αντικείμενα μπορούν να λάβουν μόνο 2 πιθανές αξίες, είτε 0 είτε 1 και ας ονομάσουμε αυτό το πρόβλημα OPT-EXT(0,1). Από την αναγωγή του OPT-EXT από το DOMINATING SET, δείξαμε ότι ακόμα και στην περίπτωση που υπάρχουν μόνο 2 πιθανές αξίες, το πρόβλημα είναι **NP**-δύσκολο, επομένως και το OPT-EXT(0,1) είναι **NP**-δύσκολο.

Μπορούμε να σκεφτούμε για το συγκεκριμένο πρόβλημα ότι έχουμε 2 κλάσεις αντικειμένων, τα «δημοφιλή» (αξίας 1) και τα «λιγότερο δημοφιλή» (αξίας 0). Στην περίπτωση της ιστοσελίδας, μπορούμε να σκεφτούμε μια απλοποιημένη εκδοχή όπου θέλουμε να τοποθετήσουμε είτε διαφημίσεις είτε δημοσιεύματα. Τα δημοσιεύματα είναι τα «δημοφιλή» και έχουν αξία 1, ενώ οι διαφημίσεις έχουν αξία 0 και αντλούν επίδραση από τα δημοσιεύματα, σε περίπτωση που είναι κοντά τους.

Για την προσέγγιση του OPT-EXT(0,1), αποδεικνύεται ότι αν έχουμε $|O_1| < |V|$ σε ένα στιγμιότυπο $\mathcal{I}_1 = (G, O_1, \nu_1)$ του OPT-EXT(0,1), μπορούμε να προσθέτουμε αντικείμενα αξίας 0 στο σύνολο O_1, δημιουργώντας ένα σύνολο O_2 με $|O_2| = |V|$, άρα και ένα στιγμιότυπο $\mathcal{I}_2 = (G, O_2, \nu_2)$ του OPT-EXT(0,1), στο οποίο κάθε ρ-προσεγγιστικός αλγόριθμος για το $\mathcal{I}_2$ είναι επίσης ρ-προσεγγιστικός για το $\mathcal{I}_1$. Έστω ότι τρέχουμε το συγκεκριμένο ρ-προσεγγιστικό αλγόριθμο στο $\mathcal{I}_2$. Η απόδειξη προκύπτει διακρίνοντας δύο περιπτώσεις:

- Παρατηρούμε ότι σε περίπτωση που έχουμε μια ανάθεση π για το $\mathcal{I}_2$ στην οποία τα αντικείμενα αξίας 0 που δε λαμβάνουν επίδραση είναι το πολύ $|O_2| - |O_1|$, τότε έστω ότι διαγράφουμε $|O_2| - |O_1|$ αντικείμενα αξίας 0 (συμπεριλαμβανομένων όλων των αντικειμένων αξίας 0 που δε λαμβάνουν επίδραση). Τότε τα αντικείμενα αξίας 0 που απομένουν όλα λαμβάνουν επίδραση, άρα έχουμε μια βέλτιστη λύση για το $\mathcal{I}_1$.

- Αν τα αντικείμενα αξίας 0 που δε λαμβάνουν επίδραση στην π είναι περισσότερα από $|O_2| - |O_1|$, τότε διαγράφοντας ακριβώς $|O_2| - |O_1|$ αντικείμενα, προκύπτει μια εφικτή ανάθεση π' για το $\mathcal{I}_1$. Θεωρώντας μια βέλτιστη ανάθεση π_1^* για το $\mathcal{I}_1$, μπορούμε να συμπληρώσουμε αντικείμενα αξίας 0 στις κορυφές v της π_1^* για τις οποίες ισχύει $\pi_1^*(v) = \perp$. Έτσι θα πάρουμε μια ανάθεση π που δεν είναι χειρότερη από την π_1^* (καθώς τα αντικείμενα αξίας 0 της π_1^* λαμβάνουν και τώρα επίδραση), αλλά που δεν είναι καλύτερη από μια βέλτιστη ανάθεση π_2^* για το $\mathcal{I}_2$ (καθώς δεν υπάρχει καλύτερη ανάθεση από τη βέλτιστη). Πολλαπλασιάζοντας με ρ, προκύπτει $\rho\mathsf{Ext}_{\pi_2^*}(G) \geq \rho\mathsf{Ext}_{\pi_1^*}(G)$ και άρα ο αλγόριθμος ειναι και ρ-προσεγγιστικός για το $\mathcal{I}_1$.

Επομένως, στη συνέχεια, όταν θεωρούμε στιγμιότυπα του προβλήματος OPT-EXT(0,1), μπορούμε να θεωρούμε (και θα θεωρούμε) χωρίς βλάβη της γενικότητας ότι το πλήθος των αντικειμένων είναι ίσο με το πλήθος των κορυφών.

1.4.2 Ένας προσεγγιστικός αλγόριθμος με σταθερό λόγο προσέγγισης

Ο αλγόριθμος 1 είναι ένας αλγόριθμος που λύνει το πρόβλημα OPT-EXT(0,1) σε σταθερό λόγο προσέγγισης, ίσο με $(e-1)/(1+e) \approx 0.46$.

Algorithm 1

Είσοδος: b, r, G με n κορυφές

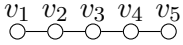
Έξοδος: Μια εφικτή λύση για το πρόβλημα AUX

1: Όλες οι κορυφές στο G είναι λευκές.
2: $sol(0) \leftarrow \emptyset$
3: **for** $t = 1$ to b **do**
4: Χρωμάτισε με μπλε χρώμα μια κορυφή v που δεν είναι ήδη μπλε, και αν είναι δυνατόν, χρωμάτισε με κόκκινο χρώμα μερικούς λευκούς γείτονες της v. Κάνε αυτό ώστε να μεγιστοποιηθεί ο αριθμός των νέων καλυπτόμενων κορυφών, υπό τον περιορισμό ότι ο συνολικός αριθμός κόκκινων κορυφών είναι το πολύ r.
5: $sol(t) \leftarrow sol(t-1) \cup \{$οι νέες κορυφές που καλύφθηκαν$\}$
6: **end for**
7: **return** $sol(b)$

Η απόδειξη έχει αρκετά μαθηματικά και περιγράφεται αναλυτικά στην ενότητα 5.2. Διαισθητικά, τοποθετούμε το αντικείμενο με τη μεγαλύτερη τιμή από αυτά που είναι διαθέσιμα στην κορυφή με τους περισσότερους "νέους" γείτονες, στους οποίους γείτονες τοποθετούμε τα αντικείμενα με τις μικρότερες τιμές από αυτά που είναι διαθέσιμα. Με τον τρόπο αυτό, από βήμα σε βήμα (δηλαδή

κάθε φορά που αναθέτουμε ένα αντικείμενο μεγάλης αξίας σε μία κορυφή), εξασφαλίζουμε η επίδραση που μπορεί να ασκήσει το συγκεκριμένο αντικείμενο να είναι η μεγαλύτερη δυνατή.

Το όριο 0.46 δεν είναι ισχυρό (tight). Αυτό σημαίνει ότι δεν έχουμε βρει παράδειγμα που υποδεικνύει ότι πράγματι η πραγματική επίδοση του αλγορίθμου για κάποιο στιγμιότυπο είναι το 0.46 της βέλτιστης λύσης. Υπάρχει όμως παράδειγμα στο οποίο η επίδοση του αλγορίθμου είναι τα 2/3 της βέλτιστης λύσης:



Στο παραπάνω παράδειγμα, αν $k = 2$, ο αλγόριθμος 1 μπορεί να αναθέσει το πρώτο αντικείμενο στην κορυφή v_3. Τότε, το δεύτερο αντικείμενο θα ανατεθεί σε μία εκ των κορυφών v_2 ή v_4 (ας θεωρήσουμε αυθαίρετα τη v_2). Τελικά, μπλε θα είναι οι κορυφές v_2, v_3 και κόκκινες οι κορυφές v_1, v_5, με αποτέλεσμα ο αλγόριθμος να επιστρέφει αποτέλεσμα 2, ενώ η βέλτιστη λύση θα ήταν να είναι μπλε οι κορυφές v_2, v_4 και κόκκινες οι v_1, v_3, v_5, άρα η βέλτιστη λύση θα έχει τιμή 3. Επομένως, για το συγκεκριμένο παράδειγμα ο αλγόριθμος έχει λόγο προσέγγισης 2/3.

Το κενό μεταξύ του $(1 - e)/(e + 1) \approx 0.46$ και του 2/3 δεν έχει καλυφθεί. Υπάρχουν τεχνικές που μπορούν να γεφυρώσουν το συγκεκριμένο όριο, αλλά προϋποτίθεται ότι οι αντίστοιχες συναρτήσεις είναι μονότονες και "submodular", αλλά αποδεικνύουμε ότι η συγκεκριμένη αντικειμενική συνάρτηση για την επίδραση δεν ανήκει σε αυτές τις κατηγορίες, επομένως δε μπορούμε να γεφυρώσουμε με αυτό τον τρόπο το χάσμα μεταξύ του $(1 - e)/(e + 1) \approx 0.46$ και του 2/3.

1.4.3 Συσχέτιση με το Μερικώς Κυρίαρχο Σύνολο

Το πρόβλημα του μερικώς κυρίαρχου συνόλου, γνωστό και ως PARTIAL DOMINATING SET, ορίζει ότι δεδομένου ενός γραφήματος G και ενός ακεραίου $t \geq 0$, να βρεθεί ένα σύνολο κορυφών του γραφήματος ώστε οι κορυφές στην ένωση των κλειστών γειτονιών των συνόλων να είναι τουλάχιστον t και ταυτόχρονα η πληθικότητα του S να είναι μέγιστη.

Θεωρούμε τώρα έναν αριθμό k. Εκτελώντας έναν αλγόριθμο που λύνει το PARTIAL DOMINATING SET n φορές, μια φορά για κάθε $t \in [1..n]$, με τον περιορισμό ότι η πληθικότητα του συνόλου S_t πρέπει να είναι το πολύ k, θεωρούμε την τιμή του t για την οποία οι καλυμμένες κορυφές μεγιστοποιούνται. Άρα έχουμε μια λύση για το OPT-EXT(0,1). Αυτό συμβαίνει καθώς τα αντικείμενα που θα ανήκουν στις κορυφές του συνόλου, θα είναι τα αντικείμενα τιμής 1, ενώ στα καλυμμένα αντικείμενα θα αναθέσουμε όσα αντικείμενα τιμής 0 μπορούμε.

Έτσι, έχουμε ότι αν ένας αλγόριθμος λύνει με ακρίβεια το πρόβλημα του PARTIAL DOMINATING SET σε χρόνο $T(n)$, τότε επειδή τον εκτελούμε n φορές, έχουμε αλγόριθμο που λύνει με ακρίβεια το πρόβλημα OPT-EXT(0,1) σε χρόνο $nT(n)$. Λόγω της έρευνας των Demaine και άλλων, προκύπτει ότι έχουμε αλγόριθμο που επιλύνει το OPT-EXT(0,1) σε χρόνο $n \cdot 3^{1.5\mathsf{tw}} n^{\mathcal{O}(1)}$ για γράφους με πάχος δέντρου το πολύ 10, ενώ λόγω της έρευνας των Fomin και άλλων, έχουμε αλγόριθμο που επιλύνει το OPT-EXT(0,1) σε χρόνο $n \cdot 2^{\mathcal{O}(\sqrt{|D_t|})} n^{\mathcal{O}(1)}$.

1.5 Το πρόβλημα OPT-EXT με γενικές αξίες αντικειμένων

Στην περίπτωση αυτή, δεν υποθέτουμε ότι $|O| = |V|$, καθώς δεν περιοριζόμαστε πλέον σε μόλις 2 αξίες αντικειμένων.

1.5.1 Γραφήματα με μέγιστο βαθμό 2

Ενώ το πρόβλημα OPT-EXT για γενικές αξίες αντικειμένων σε γραφήματα με μέγιστο βαθμό 3 και πάνω είναι **NP**-δύσκολο, μπορούμε να αποδείξουμε ότι το πρόβλημα OPT-EXT επιλύεται σε πολυωνυμικό χρόνο (ανήκει δηλαδή στην κλάση **P**) όταν ο μέγιστος βαθμός του γραφήματος είναι το πολύ 2, όταν δηλαδή το γράφημα είναι μια συλλογή από κύκλους και μονοπάτια.

Η απόδειξη είναι σύνθετη και παρουσιάζεται στην ενότητα 6.1, η ιδέα όμως είναι ότι το γράφημα μπορεί να "μετατραπεί" σε ένα βέλτιστο γράφημα, αν αναθέσουμε αρχικά τα αντικείμενα με συγκεκριμένο τρόπο και στη συνέχεια εκτελέσουμε διαδοχικές τροποποιήσεις.

1.5.2 Δέντρα-κάμπιες

Υπενθυμίζοντας ότι ένα δέντρο είναι ένα γράφημα που δεν περιέχει κύκλους, ορίζουμε ως ένα **δέντρο-κάμπια** ένα δέντρο στο οποίο αποτελείται από δύο τμήματα: το **σπόνδυλο** και τις **ακριανές κορυφές**. Στο δέντρο-κάμπια, ο σπόνδυλος είναι ένα μονοπάτι, στο οποίο όλες οι ακριανές κορυφές είναι βαθμού 1 και έχουν ένα γείτονα στο σπόνδυλο.

Αρχικά, κάνουμε την παρατήρηση ότι αν ένα γράφημα είναι μια συλλογή αστεριών (όπου ένα αστέρι είναι ένα γράφημα που έχει μία "κεντρική κορυφή" και όλες οι άλλες κορυφές είναι βαθμού 1 και συνδεδεμένες στην κεντρική κορυφή), το OPT-EXT λύνεται σε πολυωνυμικό χρόνο για το γράφημα αυτό. Αυτό συμβαίνει καθώς αναθέτοντας τα αντικείμενα μεγαλύτερης αξίας στα κέντρα των αστεριών, ξεκινώντας από το αστέρι του οποίου το κέντρο έχει τους περισσότερους γείτονες και από το αντικείμενο με τη μεγαλύτερη αξία, και αναθέτοντας στους γείτονες των κέντρων των αστεριών τα αντικείμενα μικρότερης αξίας, ξεκινώντας πάλι από το αστέρι του οποίου το κέντρο έχει τους περισσότερους γείτονες και από το αντικείμενο με τη μικρότερη αξία, η συνολική επίδραση του γραφήματος προκύπτει ότι είναι η μέγιστη δυνατή.

Αποδεικνύουμε ότι υπάρχει ένας 0.5-προσεγγιστικός αλγόριθμος για την επίλυση του OPT-EXT στην περίπτωση που το γράφημα είναι ένα δέντρο-κάμπια. Συγκεκριμένα, επειδή οι κορυφές του σπονδύλου είναι ένα μονοπάτι, άρα αν απομονώσουμε το σπόνδυλο, κάθε κορυφή έχει βαθμό το πολύ 2, γνωρίζουμε ότι απομονώνοντας το σπόνδυλο, μπορούμε να λύσουμε το πρόβλημα για τις κορυφές του σπονδύλου σε πολυωνυμικό χρόνο. Αν τώρα θεωρήσουμε τις ακριανές κορυφές, το συγκεκριμένο υπογράφημα προκύπτει ότι είναι μια συλλογή αστεριών, οπότε σύμφωνα με την παρατήρηση της προηγούμενης παραγράφου προκύπτει ότι και για αυτή την περίπτωση το OPT-EXT μπορεί να λυθεί σε πολυωνυμικό χρόνο. Επομένως, παίρνοντας την καλύτερη από τις 2 επιμέρους λύσεις, προκύπτει ότι η τελική λύση (για το συνολικό δέντρο-κάμπια) δε θα είναι χειρότερη από το μισό της καλύτερης λύσης που θεωρήσαμε. Άρα ο αλγόριθμος για το δέντρο-κάμπια είναι 0.5-προσεγγιστικός.

1.6 Πειραματικά Αποτελέσματα

Στη συγκεκριμένη ενότητα, θεωρούμε έναν αλγόριθμο για το OPT-EXT και εξετάζουμε την επίδοσή του σε σύγκριση με ένα άνω όριο της βέλτιστης λύσης.

Αρχικά, μπορούμε να ορίσουμε ένα τετριμμένο άνω όριο $T(G)$ για τη βέλτιστη λύση του OPT-EXT, ως η περίπτωση στην οποία σε ένα υποθετικό γράφημα, όλα τα αντικείμενα μπορούν να λαμβάνουν επίδραση από το αντικείμενο μέγιστης αξίας στο O. Το όριο αυτό όμως είναι τετριμμένο και αρκετά υψηλό, επομένως στις περισσότερες τοπολογίες δεν προσεγγίζει επαρκώς τη βέλτιστη λύση.

Για το λόγο αυτό, θεωρούμε ένα μη τετριμμένο άνω όριο $U(G)$ για τη βέλτιστη λύση, που θεωρεί τις κορυφές να συμμετέχουν σε μια συλλογή αστεριών. Το κέντρο κάθε αστεριού, για να υπάρχει εγγύτητα στο γράφημα για το οποίο ψάχνουμε άνω όριο της βέλτιστης λύσης, έχει βαθμό ίσο με κάθε φορά το μέγιστο βαθμό μίας κορυφής που δεν έχει ακόμα θεωρηθεί σαν κέντρο αστεριού. Με τον τρόπο αυτό, παίρνουμε ένα νέο άνω όριο $U(G)$, που είναι σημαντικά πιο κοντά στη βέλτιστη λύση, καθώς εξαρτάται και από τα χαρακτηριστικά του γραφήματος, σε αντίθεση με το $T(G|)$.

Τρέχοντας έναν άπληστο αλγόριθμο πολύ παρόμοιο με τον 1, προκύπτει ότι για αρκετά πειραματικά αποτελέσματα, ο αλγόριθμος πετυχαίνει αποτέλεσμα με απόκλιση το πολύ 5% από το $U(G)$.

Παρατηρούμε μάλιστα ότι ο αλγόριθμος έχει καλύτερη επίδοση στους πυκνούς γράφους, το οποίο θεωρούμε ότι πιθανότητα δεν είναι επειδή ο αλγόριθμος δεν τα πηγαίνει καλά στους αραιούς γράφους, αλλά επειδή στους πυκνούς γράφους τα αστέρια που απαιτούνται για να καλύψουμε ολόκληρο το γράφημα είναι λιγότερα, καθώς υπάρχουν περισσότερες κορυφές μεγαλύτερου βαθμού.

Chapter 2

Basic Notions and Preliminaries

2.1 Graphs

Let us consider a group of individuals. Some of them are friends with each other, while some of them are not. If we think of every individual as a point and draw a line between two individuals that are friends, we have modeled a simple structure of certain relations among these points. In daily life, many situations can be modeled in a similar way. If we consider the map of a country, we can model the cities as the points and the roads between the cities as the lines, by drawing a line for every two cities that are directly connected with a road. Again, we have a structure of certain relations among these points. This structure is an example of a simple graph.

We can therefore think of a graph as a data structure which models the relations between certain pairs of objects. The objects are referenced as **nodes** or **vertices**, while the lines connecting pairs of vertices are referenced as **edges**. The definition of a graph follows.

Definition 2.1.1. *A **graph** is a pair $G = (V, E)$, where V is a set of vertices and E is a set of pairs of vertices, called edges.*

A graph is **simple** when each pair of vertices belongs to $|E|$ only once. That is, when there are not multiple edges linking the same vertices. Throughout this reading, when referring to graphs, we will imply simple graphs, unless something different is stated explicitly.

An edge can be either directed or undirected. We call an edge $e = \{v, w\}$ **directed** when the edge shows a direction from vertex v towards vertex w. A *directed edge* can also be denoted as an **arc**. From now on, we will refer to undirected edges as simply *edges*, and to directed edges as *arcs*. Note the distinction in the notation of edges and arcs; we denote an edge connecting vertices v and w by $\{v, w\}$ or equivalently $\{w, v\}$, but we denote an arc from vertex v to vertex w as (v, w). In arcs, vertex v is called the *initial vertex* of the arc, while vertex w is called the *terminal vertex* of the arc. Typically, an arc is denoted with an arrow pointing to the terminal vertex of the edge. A graph that contains only edges is called an **undirected graph**, otherwise it is called a **directed graph**. Note that in the case of a directed graph with both edges and arcs, an edge $e = v, w$ can be thought as two arcs $a_1 = (v, w)$ and $a_2 = (w, v)$.

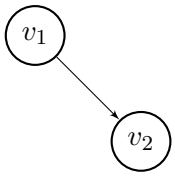


Figure 2.1: A directed edge, known as arc

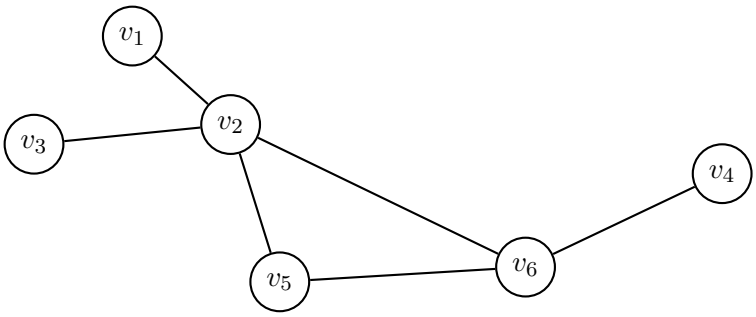


Figure 2.2: A simple, undirected, connected graph

We can think of a path in a graph as any traversal through consecutively moving through graph edges. The definition of a path follows.

Definition 2.1.2. *A **path** of length k from vertex v_1 to vertex v_k is a sequence of k vertices $v_1, v_2, ..., v_k$ for which $\{\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_{k-1}, v_k\}\} \in E$ stands.*

Let us now introduce some more definitions on concepts mentioned later in the reading.

Definition 2.1.3. *A graph is called **connected** if for any two vertices $v, w \in V$ there is a path from v to w.*

If two vertices v, w are connected with an edge, they are called *neighbors*. The definition follows.

Definition 2.1.4. *A vertex $w \in V$ is called a **neighbor** of a vertex $v \in V$ when $\{v, w\} \in E$ stands.*

If we want to denote the whole set of neighbors of a vertex v, the terms *neighborhood* and *closed neighborhood* are useful. A *neighborhood* is the set of all neighbors of v, while a **closed neighborhood** is the neighborhood of v with the vertex v included in it. The definitions follow.

Definition 2.1.5. *The **neighborhood** $\mathcal{N}(v)$ of a vertex $v \in V$ is the set S of all vertices w, for which $\{v, w\} \in E$ stands.*

Definition 2.1.6. *The **closed neighborhood** $\mathcal{N}[v]$ of a vertex $v \in V$ is the set S, formed as the union of $\mathcal{N}(v)$ and v.*

The *degree* of a vertex v is the number of neighbors of v. Formally, the degree of a vertex v is the number of elements in its neighborhood. The definition follows.

Definition 2.1.7. *The **degree** of a vertex $v \in V$ is the number of elements of $\mathcal{N}(v)$, namely $|\mathcal{N}(v)|$.*

Specifically, regarding *directed graphs*, we usually avoid using the term *degree*, since it commonly raises a question whether we include all the arcs that start from or end to the vertex in question, or only those that start from the concerned vertex. To avoid this confusion, we define two new terms, the *in-degree* and the *out-degree* of vertices in directed graphs. The definitions follow.

Definition 2.1.8. *In a directed graph $G = V, E$, the **in-degree** of a vertex $v \in V$ is defined as $|S|$, where $S := \{w \in V : (w, v) \in A\}$.*

Definition 2.1.9. *In a directed graph $G = V, E$, the **out-degree** of a vertex $v \in V$ is defined as $|S|$, where $S := \{w \in V : (v, w) \in A\}$.*

Intuitively, we can think of the *in-degree* of a vertex v as the number of arcs ending to v, while we can think of the *out-degree* of a vertex v as the number of arcs starting from v. This way, when we want to calculate the amount of arcs *incident* to v (that is, starting from or ending to v) we just sum the in-degree and the out-degree of v.

Let us now define some terms regarding subgraphs. Intuitively, we define a graph G_1 to be a *subgraph* of G_2 when G_1 contains some of the vertices and some of the edges of G_2 (possibly all or none). We define a graph G_1 to be an *induced subgraph* of G_2 when G_1 contains some of the vertices of G_2, and all of the edges of G_2 that have vertices of G_1 as endpoints are also edges of G_1. We define a graph G_1 to be a *spanning subgraph* of G_2 when G_1 and G_2 have exactly the same vertices, and G_1 contains some of the edges of G_2. The formal definitions follow.

Definition 2.1.10. *A graph $G_1 = (V_1, E_1)$ is a **subgraph** of a graph $G_2 = (V_2, E_2)$ when $V_1 \subseteq V_2$ or $E_1 \subseteq E_2$ (or both).*

Definition 2.1.11. *A graph $G_1 = (V_1, E_1)$ is an **induced subgraph** of a graph $G_2 = (V_2, E_2)$ when $V_1 \subseteq V_2$, $E_1 \subseteq E_2$ and $\forall e = \{v_a, v_b\} \in E_2$ so that $v_a, v_b \in V_1$: $e \in E_1$.*

Definition 2.1.12. *A graph $G_1 = (V_1, E_1)$ is a **spanning subgraph** of a graph $G_2 = (V_2, E_2)$ when $V_1 \equiv V_2$ and $E_1 \subseteq E_2$.*

Similarly to the definition of subgraphs, the definition of a *minor* graph follows.

Definition 2.1.13. *A graph $G_1 = (V_1, E_1)$ is a **minor** of a graph $G_2 = (V_2, E_2)$ if G_1 can be created by deleting edges, vertices and contracting edges of G_2. A **contraction** of an edge v_1, v_2 means to delete the edge $\{v_1, v_2\}$ and to merge the vertices v_1 and v_2. A **merge** of two vertices v_1 and v_2 is to create a new vertex v, with all edges $\{v_1, w\} \in E$ or edges $\{v_2, w\} \in E$, for any w being replaced by new edges $\{v, w\}$.*

In Figure 2.3, we can see a graph G before the *contraction* of the edge $\{v_2, v_5\}$. In Figure 2.4, a new graph G' has been formed by contracting the edge $\{v_2, v_5\}$ in G. Observe that

vertices v_2 and v_5 have been *merged* into a new vertex v', who has as neighbors every vertex in the set $\mathcal{N}(v_2) \cup \mathcal{N}(v_5)$, namely here vertices v_1, v_3, v_6. Therefore, according to Definition 2.1.13, graph G' is a *minor* of graph G, since G' has been formed from G by performing an edge contraction.

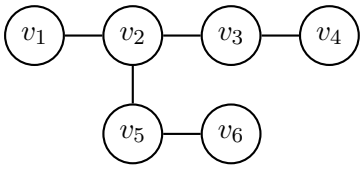


Figure 2.3: Graph G, before the contraction of edge $\{v_2, v_5\}$

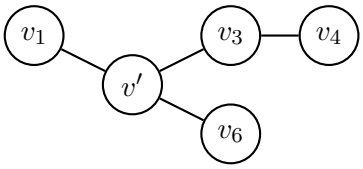


Figure 2.4: Graph G', after the contraction of edge $\{v_2, v_5\}$

In an undirected graph, we say that two vertices v, w participate in a cycle, when there are two different *paths* (i.e. with all of their vertices different except from v and w that start from v and end to w. The definition follows.

Definition 2.1.14. *In an undirected graph $G = (V, E)$, two vertices $v, w \in V$ participate in a* ***cycle*** *if there are at least two paths P_1, P_2 from v to w, that have only v and w as common vertices. We then say that the cycle contains all the vertices that belong in P_1 and P_2. We can denote a cycle as a path, starting an ending at the same vertex. The size of a cycle C is the number of edges it contains and is denoted by $|C|$. A cycle has, by definition, a size of at least 3.*

In Figure 2.5, we observe the following 8 cycles:

- $C_1 = \{(v_1, v_2), (v_2, v_3), (v_3, v_7), (v_7, v_1)\}$ (size 4)
- $C_2 = \{(v_2, v_4), (v_4, v_5), (v_5, v_2)\}$ (size 3)
- $C_3 = \{(v_2, v_4), (vs_4, v_6), (v_6, v_2)\}$ (size 3)
- $C_4 = \{(v_2, v_5), (v_5, v_6), (v_6, v_2)\}$ (size 3)
- $C_5 = \{(v_4, v_5), (v_5, v_6), (v_6, v_4)\}$ (size 3)
- $C_6 = \{(v_2, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_2)\}$ (size 4)
- $C_7 = \{(v_2, v_4), (v_4, v_6), (v_6, v_5), (v_5, v_2)\}$ (size 4)
- $C_8 = \{(v_2, v_6), (v_6, v_4), (v_4, v_5), (v_5, v_2)\}$ (size 4)

Therefore, we can see that while for two cycles C_1, C_2 the vertices can be the same, the edges can be different, which means that the two cycles might be different even with the same vertices.

Cycles are crucial for graph theory and whole classes or graphs are formed based on cycles or their absence. A *tree* is a subcategory of graphs, denoting the graphs that do not contain cycles. *Trees* are a very common data structure in graph theory. The definition for a tree follows.

Definition 2.1.15. *A graph $G = (V, E)$ is a* ***tree*** *if it does not contain any cycles.*

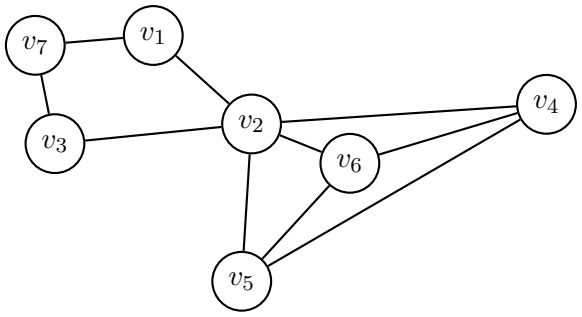


Figure 2.5: An undirected graph with 8 cycles

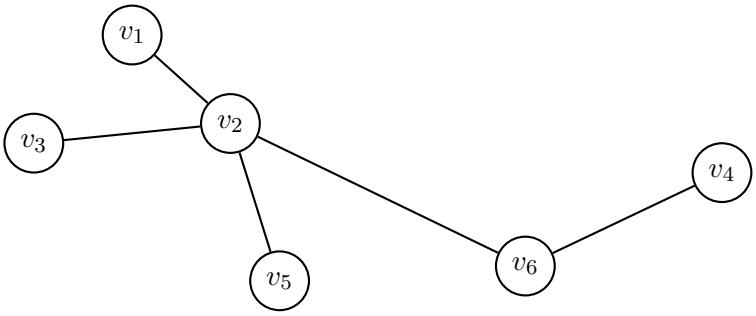


Figure 2.6: A tree

In figure 2.6 we see an example of a tree.

Let us define a special class of edge, related to cycles named *chord*. Similarly to geometry, a chord for a cycle is an edge that is not part of the cycle, but connects two vertices of the cycle. The definition for the chord follows.

Definition 2.1.16. *A **chord** c for a cycle C in a graph $G = (V, E)$ is an edge $c = \{v_1, v_2\} \in E$ connecting two vertices $v_1, v_2 \in C$, so that $c \notin C$.*

For example, in figure 2.5, the edge $\{v_2, v_6\}$ is a *chord* for the cycle C_7, since cycle C_7 does not contain the edge $\{v_2, v_6\}$, but vertices v_2 and v_6 are both included in cycle C_7. Intuitively, we can think of a chord as an edge connecting two vertices in a cycle that are not adjacent to each other in the cycle.

Let us now define a new class of graphs, called *chordal graphs*. Intuitively, chordal graphs are simple graphs that have as many chords as possible. The definition for a chordal graph follows.

Definition 2.1.17. *A graph $G = (V, E)$ is a **chordal graph** when every cycle C of G with $|C| \geq 4$ has a chord.*

For instance, we can observe that the graph in figure 2.5 is not chordal, since C_1 does not have a chord. The graph of figure 2.5 would become chordal if we added the edge $\{v_1, v_3\}$ (or the edge $\{v_2, v_4\}$). The definition of a *chordal completion* follows.

Definition 2.1.18. *A **chordal completion** for a graph $G = (V, E)$ that is not chordal is a chordal graph $G' = (V, E')$ that has G as a spanning subgraph.*

Since Definition 2.1.18 may allow "irrelevant" edges to be added (i.e. edges that do not contribute to the chordal completion) towards the creation of G', what really matters is the *minimum chordal completion*, which is the chordal completion by adding only as many edges as necessary. The definition follows.

Definition 2.1.19. *A **minimum chordal completion** for a graph $G = (V, E)$ that is not chordal is a chordal graph $G' = (V, E')$ that has G as a spanning subgraph and $|E'| - |E|$ is minimized.*

For example, in Figure 2.5, a chordal completion would be to add edges $\{v_1, v_4\}$ and $\{v_1, v_3\}$, but although the resulting graph would be chordal, this chordal completion would not be minimum, since we can get a chordal graph by adding only one edge (either $\{v_1, v_3\}$ or $\{v_2, v_7\}$).

A *clique* is a subset of vertices in a graph in which every vertex is connected to every other vertex. The defintion for the clique follows.

Definition 2.1.20. *A **clique** in a graph $G = (V, E)$ is a set of vertices $S \subseteq V$ for which $\forall v \in S : S \subseteq \mathcal{N}[v]$.*

In this manner, we define the *maximum clique* in a graph to be a clique, so that there is no clique with a higher number of vertices. The definition follows.

Definition 2.1.21. *A **maximum clique** in a graph $G = (V, E)$ is a clique $S \subseteq V$ for which $|S|$ is maximum.*

Let us now define a different type of chordal completion $\mathcal{C}$, namely a chordal completion that minimizes the size of the maximum clique in the resulting chordal graph. The size of the maximum clique in $\mathcal{C}$ can be defined as the *treewidth* of the graph. In other words, the *treewidth* of the graph is the infimum of the maximum clique in all possible [minimum] chordal completions of a graph. The definition of the treewidth follows.

Definition 2.1.22. *Consider a graph $G = (V, E)$ and a chordal completion $\mathcal{C}$ that minimizes the size of maximum clique in G. Consider the maximum clique in $\mathcal{C}$ to be $\mathcal{S}$. The **treewidth** of G is equal to $|\mathcal{S}|$ and is denoted by $\mathsf{tw}(G)$.*

Much study has been performed on treewidth of graphs as a property and more about treewidth will be mentioned in the next sections.

Let us now talk about a different category of graphs, namely graphs that can be represented in a two-dimensional plane. Let us define such graphs as *planar* graphs. The definition follows.

Definition 2.1.23. *A **planar graph** is a graph $G = (V, E)$ that can be drawn in such a way that the edges intersect only at their endpoints.*

The graph in Figure 2.3 is planar, while the graph in Figure 2.7 is not planar.

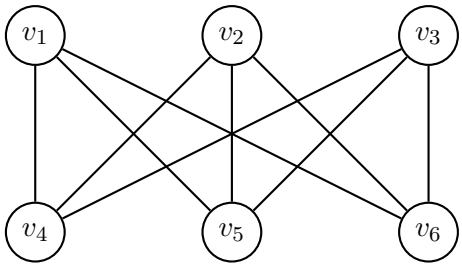


Figure 2.7: A non planar graph

If a graph G is not planar for only one vertex (i.e. if one vertex is removed from G along with the edges incident to it, G becomes planar), G is called an *apex* graph. The definition for an apex graph follows.

Definition 2.1.24. *A graph $G = (V, E)$ is an **apex** graph, if there exists a vertex $v \in V$, for which there is a planar induced subgraph $G' = (V \setminus v, E')$, where E' is E without the edges incident to v.*

By characterizing a graph G_1 as **forbidden**, we denote that the presence of the graph G_1 as subgraph or minor graph of a graph G_2, means that G_2 cannot have certain properties, forbidden by the existence of graph G_1 as a subgraph. The most simple example is that a graph G_2 cannot be a tree if it contains a cycle as a subgraph. In fact, the graph of Figure 2.7 is a forbidden graph for planar graphs, according to **Kuratowski's theorem** [38].

Let us now define the notion of *matching* vertices. Intuitively, we can think of *matching* vertices as pairing vertices in groups of two, which are already neighbors of each other. In a *matching*, however, we consider that a vertex belongs to at most one group of two. The formal definition follows.

Definition 2.1.25. *A **matching** in a graph $G = (V, E)$ is a set of edges M, for which every vertex $v \in V$ is incident to at most one edge of M.*

Based on the definition for matching, we can define a maximum matching as a matching with the maximum possible number of edges. Note that a maximum matching may not be unique for a given graph G. The definition follows.

Definition 2.1.26. *A **maximum matching** in a graph $G = (V, E)$ is a matching M for which $|M|$ is maximum.*

Maximum matching should not be confused with *maximal matching*, which is a matching for which the addition of any edge not in the matching leads in a set of edges that is not a matching. The definition of a *maximal matching* follows.

Definition 2.1.27. *A **maximal matching** in a graph $G = (V, E)$ is a matching M, so that for any edge $e \in E \setminus M$, the set $M \cup \{e\}$ is not a matching.*

While it is clear that a maximum matching is always maximal, it should be noted that a maximal matching may not be maximum. Considering a simple path of 4 vertices $\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}$,

note that the edge $\{v_2, v_3\}$ alone is a maximal matching, but not a maximum one, since the set of edges $\{\{v_1, v_2\}, \{v_3, v_4\}\}$ is a matching of larger size.

2.2 Algorithms and Complexity

Let us consider the problem of trying to simplify a fraction. The process we are following is standard: A fraction consists of a nominator n and a denominator d. If the fraction is not yet simplified, it means that the greatest common divisor (**GCD** from now on) of n and d is larger than 1. In order words, that n and d are not *coprime*. We can note that if we divide n and d with their GCD, the new numbers we will get will be coprime, and the fraction will be simplified.

We can therefore see that the problem is the same as "finding the gcd of n and d". A trivial way to solve this problem would be to examine all numbers from 1 to the minimum of n and d, and find the largest number x that is a divisor of both n and d, or, in order words, that satisfies $n \bmod x = 0$ and $d \bmod x = 0$. We can formulate this procedure in the following steps (where "$\leftarrow$" indicates an assignment of the value on the right to the variable on the left):

Procedure 2 A trivial way to find the greatest common divisor of two numbers

Input: Integers n (nominator) and d (denominator)
Output: Integers a and b so that $n/d = a/b$ and $gcd(a, b) = 1$

1: $minimum\_number \leftarrow min(n, d)$
2: **for** $t = 2$ to $minimum\_number$ **do**
3: **if** $n \bmod t = 0$ **and** $d \bmod t = 0$ **then**
4: $x \leftarrow t$
5: **end if**
6: **end for**
7: $a \leftarrow n/x$
8: $b \leftarrow d/x$
9: **return** a, b

This procedure is precisely an algorithm. The definition of an algorithm follows.

Definition 2.2.1. *An **algorithm** is a finite sequence of well-defined, computer-implementable instructions, typically to solve a class of problems or to perform a computation.*

Typically, computers can execute about 10^8 instructions per second. This means that while Procedure 1 can be useful for numbers up to $10^6 - 10^7$, it is not fast enough for numbers much larger (such as 10^{13} or 10^{16}). This is a matter of **computational complexity**. Procedure 1 uses linearly as many instructions as the minimum of n and d, since it is going to calculate if all numbers up to $min(n, d)$ are divisors of n and d. Therefore, Procedure 1 has **linear complexity** in the worst case.

In order to upper bound the worst-case complexity of an algorithm, we use **big O notation**.

By using big O notation, the running time of an algorithm is bounded according to a function with the input size as a parameter. Therefore, execution time that increases linearly as the input size of $min(n, d)$ increases is denoted $O(min(n, d))$ and that is the big O notation for the computational complexity of Procedure 1.

In order to be able to execute quickly Procedure 1 for input sizes larger than 10^7, we need to find an algorithm that has significantly lower computational complexity. Since we would like to calculate numbers up to $10^{16} - 10^{18}$ with up to 10^8 instructions, we need the computational complexity to be better than linear, i.e. *sublinear*. Let us introduce the **Euclidean algorithm** for the GCD of two numbers, which is the following:

Algorithm 3 Euclidean algorithm for the greatest common divisor of two numbers

Input: Integers n and d

Output: Integers a and b so that $n/d = a/b$ and $gcd(a, b) = 1$

1: $a \leftarrow n$
2: $b \leftarrow d$
3: **while** $b \neq 0$ **do**
4: swap(a, b)
5: $b \leftarrow b \bmod a$
6: **end while**
7: **return** a, b

Algorithm 2 provides the correct result, with a computational complexity much smaller than Procedure 1, equal to $O(\log min(n, d))$. The difference in the execution time is so high, that the execution of Algorithm 2 for numbers up to 10^{18} takes the same time with the execution of Procedure 1 for numbers up to 60.

We consider problems to be able to be solved *efficiently* if they are available to be solved in **polynomial time**. We consider an algorithm able to be solved in *polynomial time* if its running time is upper bounded by a polynomial expression in the size of the input for the algorithm. Examples of polynomial time complexities in terms of big O notation are:

- $O(n^3)$
- $O(n \cdot \sqrt{n})$
- $O(n^{3.95} log(n))$

However, there are problems for which no efficient algorithms have been found yet. We call these problems *non-deterministic polynomial time complete*, or simply **NP**-complete.

Chapter 3

State of the Art

In this paper, we are going to consider the concept of externality. Intuitively, externality describes a situation where the value of an object does not depend entirely on it, but also from the way that the other objects surround it. Let us describe notions of externality already studied in certain study fields.

3.1 Economics

Research has been conducted regarding the externality of an object in economics. We will mention a few examples.

In [44], a model of externality regarding communications is described, such that the external utility that a subscriber derives from a communications is increasing as more users join the system. Starting with explaining the theory of demand, several econimic analyses are conducted, all taking into consideration the fact that the utility function of an existing subscriber never decreases with the addition of new subscribers, with no existing subscribers dropping from the service.

In [33], three cases of positive consumption externalities are mentioned, namely:

- The number of purchasers of the product might have a direct physical effect on the consumption externalities derived from the product. Presenting the example of the telephone, the utility derived from the purchase of a telephone (and consequently, the connection of the user in the telephone network) is clearly dependent on the number of other households and businesses that have a telephone. This can be explained that in case many households and businesses already have a telephone number, the purchase of the telephone will be severely more helpful to the user, as in that way, they can increase the total convenience (or, *externality* they will derive from the network.

- Aside from the direct effects, there can be indirect effects that increase consumption externalities. Let us say that someone purchases a computer C. Let us consider the case in which 10 persons buy C and the case in which $10,000$ persons buy C. It is clear that in the second case, the number of peripherals and the variety of software created for C and

adjusted to function properly on C will be significantly larger. This hardware-software example may also apply to video games, video players and phonographic equipment.

- It is possible that the quality of service and technical support after the purchase of goods depends on the experience and the size of the service network, which may directly or indirectly depend on the number of units sold, or the number of subscribers in a network. Taking the automobile market as an instance, the sales of foreign manufacturers were initially hindered by the lack of experience and the thinness of the service networks that existed for the new or the less popular brands.

In all of these cases, we can see that the utility that the users of a unit or service derives is enormously dependent upon the number of the other users in the same "network". [33] moves on to develop a model of oligopoly to analyze markets based on the observation of positive consumption externalities, which shows the impact that externality studies have on academic research.

In [48], a model with linear externalities is analyzed, in which the welfare of a certain agent is linearly dependent to the consumption of another agent. It is being assumed that for two agents i, j, the effect that the consumption of money of j, let it be x_j, is a constant proportion of x_j on the welfare of agent i. Note that this does not imply that j is the only agent that affects the welfare of agent i. The paper moves on to further analyze the matters from an economic aspect.

3.2 Computer Science

For computer science, externality appears in many problems of interest, including *matchings*, *auctions*, and *fair allocations of divisible or indivisible goods*. We will mention some significant, to our view, related work for this matter conducted in Computer Science.

Externality has been used in matchings from Brânzei *et al.* [7] in a way that is similar to the notion we desire to deliver. Explicitly, in [7], a *matching game with additive externalities* is defined, in a way that the utility of an agent is the sum of the values it receives from matches it participates it. That is, assigning pairs of agents z_1, z_2 from two disjoint sets M and W, a *match* is being formed, from which both z_1 and z_2 derive positive utility, defined as *externality*.

In auctions, there are many papers involving externality. For advertisement auctions, Ghosh and Mahdian [21] suggest that the performance of an item is dependent on which other items are selected and displayed simultaneously. In this case, it is possible (and often happens) that the externality derived is negative, since the auction might be a *single-item auction*, and a good advertisement displayed next to a weaker one might exert a negative externality to the weak advertisement, discouraging people to opt for the product or service advertised from the weak advertisement. Negative externality may also be derived from specific parts which are worse than others, making it more difficult for the page to yield positive externality from its position, comparing it to the position of the other advertisements. The category in

which positioning matters for the value of the item is regarded as *position auctions*. Fotakis *et al.* [17] used a model for externalities among advertisers used in single-keyword auctions in sponsored search. In this model, is it assumed that externalities can be both positive or negative between any pair of advertisers. In this paper, multiple computational results are computed regarding the Winner Determination problem for Social Welfare maximization, eventually evaluating the Generalized Second Price mechanism in presence of externalities. There exist more examples for *position auctions* [4, 32].

Regarding *resource allocation*, it is often supposed that the utility of an agent depends solely on the share allocated to them, usually in respect to a function that values the share the user gets. When externalities are included, the model is different, since the utility of the agent is (often heavily) influenced by the shares of other agents as well. Seddighin *et al.* [45] considered the problem of fair allocation of certain indivisible objects, with the aim being to satisfy an adapted notion of the maximin-share criterion. In both [45] and our model, there exists a social graph whose vertices are the agents and positive externalities are exerted/derived along the edges/arcs.

The problem that will be presented through our model is named OPT-EXT and is analyzed further in the following chapter. OPT-EXT bears a stark resemblance to models that attempt to maximize the influence in social networks, such as that of Kempe *et al.* [34]. In influence maximization problems, the goal is to maximize the adoption of a new product through a social network. This process happens in rounds; in every round, a vertex adopts the product with a probability proportionate to the fraction of its neighbors that have adopted it. We can think of it as, when some friends of an individual have a certain interest, it is more likely for the individual to develop it as well. The goal is to compute a set of k initial adopters, so as to maximize the expected final number of adopters. Such problems have significant applications to marketing and pricing in social networks and they have received widespread attentions, as can be observed in [28, 3, 18] and their references. OPT-EXT does not use probabilities, but uses deterministic externalities instead that occur during a single round and deals with the solution regarding different object valuations. A big difference between the two models is that while in influence maximization models, the objective optimization function is a *monotone submodular* one (as defined in section 2.1), we will observe in section 5.2 that this is not necessarily the case for OPT-EXT.

A more specific case of OPT-EXT will be presented in our model as OPT-EXT(0,1), regarding objects valued only with valuations 0 or 1. This model belongs in the family of covering problems, in the notion that vertices receiving an object of value 1 "cover" their neighbors of value 0 by exerting positive externality to them. The goal of the problem is to maximize the number of "covered" vertices with objects of valuations 0. DOMINATING SET and MAX COVERAGE are two typical and related coverage problems. In DOMINATING SET, a vertex v in the set covers ("dominates") itself and its neighbors, and the goal is to find a set of minimum cardinality that "dominates" every vertex in the graph. Related papers are [19] and [10]. The variant of PARTIAL DOMINATING SET for DOMINATING SET exists, where the goal is to dominate at least t vertices by using the minimum number of vertices [13, 15]. The DOMINATING SET and PARTIAL DOMINATING SET problems are also explained later. In MAX COVERAGE, we are given a universe U of objects and a collection $\mathcal{C}$ of sets. The goal

is to cover the maximum possible number of objects in U by using at most k sets of $\mathcal{C}$. It can be shown that this problem is **NP**-hard, and can be approximated with a ratio of $(1-(1-1/k)^k) > 1-1/e$ [30].

The model we present is also related with the problems studied in [5, 9]. While in these articles, the allocation of goods in the vertices of a graph is studied, the objective and the motivation is completely different to that of our model.

Chapter 4

Graph Externality Model

4.1 Motivation

Let us consider a TV channel director, who has to arrange the time slots for the programs to broadcast. It is clear that if show A, which is popular, precedes show B, which is less popular, some people who will watch show A are also going to watch show B. Compared to the case where show B succeeds show C, which is much less popular than show A, it is clear that many more viewers will view show B in the case it broadcasts right after show A. The same can be said in case show B preceded show A; it is clear that people waiting to watch show A, have increased probability to watch show B as well while waiting. Therefore, the TV channel director could benefit from this and schedule the shows so that the less popular show B gets some "external" audience from the more popular show A, by putting it right before or right after show A.

Similarly, we can imagine how certain articles in a web page or a newspaper are arranged. Topology is crucial there, since posts next to popular posts are much more commonly viewed. Therefore, if the website has an interest to promote certain articles to be viewed, the owner can put the articles next to the popular ones, so that they gain some "external" clicks. It is natural that when the reader views a specific post, their interest for the neighboring contents might be increased.

Since "external" factors might be crucial to the behavior of the public, we introduce a general model that, by modeling an instance as a graph and the open slots and positions as the vertices of the graph, we attempt to approach the best possible object allocation so that the total "external" value for the graph is maximized.

4.2 Definitions and Problem Description

Let us consider a set of m objects (which can also be called goods), denoted as $O = o_1, ..., o_m$. Each object has a valuation, which corresponds to the intrinsic value the object has. The valuation can only affect the neighbors in a positive manner, therefore we consider it to be non-negative. Therefore, each object $o_i \in O$ has a **valuation** $\nu(o_i)$. We also consider a

graph topology, to the vertices of which the objects are assigned. Formally, there exists an undirected graph $G = (V, E)$ with n vertices, denoted as $v_1, v_2, ..., v_n$. Since the vertices must be enough to be able to have all the objects assigned to them, $n = |V| \geq |O| = m$ must hold.

It is important to note that every vertex can take at most one object, that is, as well as that every object must be placed on exactly one vertex. We define an **allocation** as a function $\pi : V \to O \cup \{\perp\}$, in which every object has exactly one ancestor, namely, that every vertex is allocated to at most one object, and by $pi(v) = \{\perp\}$ we denote that vertex v does not receive any object.

As stated in chapter 1, two vertices i, j which belong to V are *neighbors* if they are connected by an edge, namely if $\{i, j\} \in E$. The *neighborhood* of a vertex v is denoted with $\mathcal{N}(v)$. The *closed neighborhood* of a vertex v is denoted with $\mathcal{N}[v]$. Note that $\mathcal{N}[v] := N(v) \cup v$.

In the main model, we consider that a vertex can *derive* externality from at most one neighbor. Although the externality is derived depending on the valuations of the objects, the theoretical point concerns the externalities derived from the vertices to which the objects deriving externality are allocated. We say that a vertex v derives externality from a vertex w if both vertices receive objects, they are neighbors, the object allocated to w has a greater value than the object allocated to v, and the object allocated to w is the object with the greatest externality in the neighborhood of v. Formally, v **derives externality** from w if the following conditions are met:

1. $\pi(v) \neq \perp$
2. $\pi(w) \neq \perp$
3. $\{v, w\} \in E$
4. $\nu(\pi(v)) < \nu(\pi(w))$
5. $\forall u \in \mathcal{N}(v) : \nu(\pi(u)) \leq \nu(\pi(w))$

Conditions 1 and 2 suggest that vertices v and w must both have objects, so that the object allocated to v be able to derive externality from the object allocated to w. Condition 3 suggests that v and w must be connected with an edge. Condition 4 suggests that v will derive externality from w and not vice versa, while condition 5 suggests that w has the object with the highest valuation in the neighborhood of v allocated to it.

As a corollary, if there are no objects allocated to a neighbor of a vertex v, then v does not derive externality. In case the valuation of the object allocated to a vertex v (the respective object valuation is $\nu(\pi(v))$) is higher or equal to every neighbor of v that has an object allocated to it, v does not derive externality. It can be inferred that v derives externality only if an object has been allocated to v and the externality is derived only from the neighbor w that has the object with the highest valuation in the neighborhood of v. In this case, vertex v derives externality equal to $\nu(\pi(w)) - \nu(\pi(v))$, that is, the difference of valuation

$\nu(\pi(v))$ from valuation $\nu(\pi(w))$. The definition for the externality a vertex v derives under an allocation π follows.

Definition 4.2.1. *The **externality** a vertex v derives under an allocation π, denoted as $ext_\pi(v)$ is:*

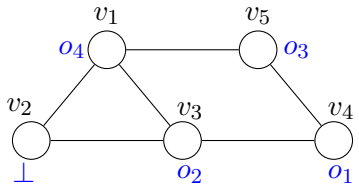
- *if $\pi(v) = \perp$ or $\forall u \in \mathcal{N}(v)$: $\pi(u) = \perp$, the externality derived is 0*
- *otherwise, letting w be the vertex for which $\forall u \in \mathcal{N}[v] : \nu(\pi(u)) \leq \nu(\pi(w))$, the externality derived is $\nu(\pi(w)) - \nu(\pi(v))$*

Finally, to define the *graph externality* of the graph $G = (V, E)$ under an allocation π, we sum the externalities of all vertices in V. The definition follows.

Definition 4.2.2. *The **graph externality** of a graph $G = (V, E)$ under an allocation π, denoted as $\mathsf{Ext}_\pi(G)$, is $\sum_{v \in V} ext_\pi(v)$.*

We will now provide two examples to demonstrate the concepts of *externality* and *graph externality*.

Consider the following instance with 4 objects.



$\nu(o_1) = 1$
$\nu(o_2) = 4$
$\nu(o_3) = 5$
$\nu(o_4) = 9$

Let the allocation $\pi = (o_4, \perp, o_2, o_1, o_3)$. We have

- $ext_\pi(v_1) = \nu(o_4) - \nu(o_4) = 0$
- $ext_\pi(v_2) = 0$
- $ext_\pi(v_3) = \nu(o_4) - \nu(o_2) = 5$
- $ext_\pi(v_4) = \nu(o_3) - \nu(o_1) = 4$
- $ext_\pi(v_5) = \nu(o_4) - \nu(o_3) = 4$

.

Therefore, $\mathsf{Ext}_\pi(G) = ext_\pi(v_1)+ext_\pi(v_2)+ext_\pi(v_3)+ext_\pi(v_4)+ext_\pi(v_5) = 0+0+5+4+4 = 13$.

The inverse allocation function under an allocation π is used to specify the *location* of an object o, that is, the vertex in which the object has been allocated under the allocation π. The definition follows.

Definition 4.2.3. *The **location** of an object o under an allocation π is the vertex to which o has been allocated and is denoted by $\pi^{-1}(o)$.*

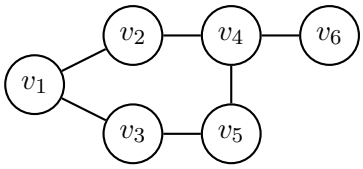


Figure 4.1: A graph with 6 vertices

Since under an allocation we force all objects to be allocated to a vertex, it follows that $\pi^{-1} : O \to V$. We can also define the notion of externality *along an edge*. Externality exists along an edge when the edge is connecting two vertices v and w, in which one derives externality from the other. The definition follows.

Definition 4.2.4. *There is* ***externality along the edge*** $\{v, w\}$ *if* $\{v, w\} \in E$ *and* v *derives externality from* w*.*

Having defined all the useful concepts, we can now define the problem OPT-EXT, which is based on finding the allocation that maximizes the graph externality of a graph, with given objects and graph topology.

Definition 4.2.5. OPT-EXT*: Given a graph* $G = (V, E)$*, a set of objects* O *and their valuations* ν*, find the allocation* π *that maximizes* $\mathsf{Ext}_\pi(G)$*.*

We define an instance of OPT-EXT as (G, O, ν), where G is the graph, O is the set of objects and ν are the respective valuations of the objects.

4.3 Hardness of OPT-EXT

Before proving the hardness of OPT-EXT, we will define and examine the related DOMINATING SET problem. Let us firstly describe a dominating set. We say that a set D of vertices of a graph G is a *dominating set* (of vertices), when every vertex of G which is not in D has at least one neighbor in D. In other words, a dominating set D of a graph G is a set, so that every vertex of G is either in D, or neighbors a vertex in D. The formal definition for a dominating set follows.

Definition 4.3.1. *A* ***dominating set*** D *in a graph* $G = (V, E)$ *is a set* $D \subseteq V$ *so that every* $v \in V \setminus D$ *has a neighbor in* D*.*

An example of a dominating set is demonstrated below.

Example 4.3.2. *Consider the following instance:*

Let us pick the set $D_1 = \{v_1, v_5\}$*. The black vertices in the following figure are the ones included in the set* D_1*, while the striped vertices are the ones that do not belong in* D_1*, but neighbor at least one vertex in* D_1*:*

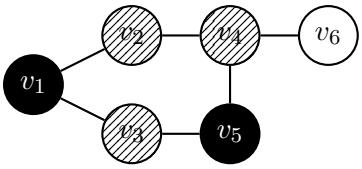


Figure 4.2: The vertices dominated by D_1

*We observe that since v_6 does not neighbor any vertex in D_1 and is not included in it, then D_1 is **not** a dominating set. Now, let us pick the set $D_2 = \{v_3, v_4\}$:*

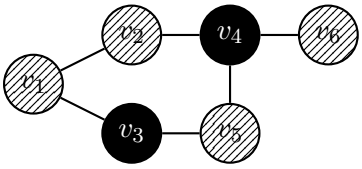


Figure 4.3: The vertices dominated by D_2

*We see that every vertex in V either belongs in D_2, or neighbors a vertex in D_2. Therefore, D_2 is a **dominating set** for G.*

In example 4.3.2, we can see that D_2 is also a **minimum size** dominating set (there is no dominating set of size 1, since there is no vertex that neighbors every other vertex). However, there can also be dominating sets of size more than 2, such as $D_3 = \{v_1, v_4, v_5\}$ (since v_2 neighbors v_1 and v_4, v_3 neighbors v_1 and v_5 and v_6 neighbors v_4). Even $D_4 = V$ is a dominating set, since every vertex in V is in D_4, so every vertex is *dominated*.

Since finding any dominating set is trivial, the DOMINATING SET problem is focused on finding a dominating set with at most k vertices, for an integer k. The definition of the DOMINATING SET problem follows.

Definition 4.3.3. DOMINATING SET*: Given a graph $G = (V, E)$ and an integer k, determine if a dominating set D where $|D| \leq k$ exists.*

We define an instance of DOMINATING SET as (G, k), where G is the graph and k is the integer, so that we check if a dominating set D with $|D| \leq k$ exists.

It has been shown that DOMINATING SET is **NP**-complete, even in the case where G is planar graph with a maximum vertex degree 3 [19], as well as in bipartite graphs and split graphs [10]. We will now use the **NP**-completeness of DOMINATING SET to prove the hardness of OPT-EXT.

Proposition 1. *Given an instance (G, O, ν) of* OPT-EXT *and a number k, determining if an allocation π so that $\mathsf{Ext}_\pi(G) \geq k$ exists is **NP**-complete, even when the valuations are 0 or 1.*

Proof. Let us consider an instance of DOMINATING SET (G, k) and have k objects with valuation 1 and $|V| - k$ objects with valuation 0. We will prove that there exists an allocation π

so that $\mathsf{Ext}_\pi(G) \geq |V| - k$ iff G has a dominating set of size k. In other words, we will show that OPT-EXT is an instance of the known **NP**-complete DOMINATING SET problem.

Let D be a dominating set of size k of G. We assign the k objects of valuation 1 to the vertices of D. Thus, the remaining $|V| - k$ vertices will be assigned objects with valuation 0. For every vertex $v \in D$, the valuation allocated to v will be 1, therefore v will have the highest valuation in $\mathcal{N}[v]$, meaning that $ext_\pi(v) = 0, \forall v \in D$. For every vertex $v \notin D$, the valuation allocated to v will be 0. Since D is a dominating set, every $v \notin D$ has at least one neighbor $w \in D$ and since $w \in D$, we get $\pi(w) = 1$. This means that $\forall v \notin D$: $\exists w$ s.t. v derives externality from w. Since $\pi(w) = 1$ and $\pi(v) = 0$, we get $ext_\pi(v) = 1$, $\forall v \notin D$. Therefore, for every vertex $v \in V$, we have:

- $ext_\pi(v) = 0$, if $v \in D$
- $ext_\pi(v) = 1$, if $v \notin D$

Therefore, in this case, we get $\mathsf{Ext}_\pi(G) \geq |V| - k$.

Conversely, suppose there exists an allocation π so that $\mathsf{Ext}_\pi(G) \geq |V| - k$. Let D be the vertex set, so that $\forall v \in D : \pi(v) = 1$. We can observe that the remaining vertices are at most $|V| - k$, while we know that $\mathsf{Ext}_\pi(G) \geq |V| - k$. Since we have considered valuations of 0 or 1, for every vertex $v \in V$ we have $ext_\pi(V) \leq 1$. Therefore, every vertex $v \in V$ can contribute up to 1 to $\mathsf{Ext}_\pi(G)$. Since $\mathsf{Ext}_\pi(G) \geq |V| - k$ and already k vertices have a valuation of 1, therefore an externality of 0, it follows that each of the remaining $|V| - k$ vertices must have an externality of 1. Since the remaining $|V| - k$ vertices are the vertices not in D and they all have an externality of 1, each one of the vertices $v \notin D$ is neighboring a vertex in D. Therefore, D is a dominating set.

■

Because of the hardness of OPT-EXT, we tackle the problem through approximate solutions and certain variations. For an instance (G, O, ν) of OPT-EXT, we consider an *optimal allocation* to be the allocation that maximizes the graph externality. We consider the graph externality yielded by the *optimal allocation* to be the *optimal solution* for the instance. The definitions follow.

Definition 4.3.4. *For an instance (G, O, ν) of* OPT-EXT*, an **optimal allocation**, denoted by π^*, is the allocation so that for every possible allocation π, we get $\mathsf{Ext}_{\pi^*}(G) \geq \mathsf{Ext}_\pi(G)$.*

Definition 4.3.5. *For an instance (G, O, ν) of* OPT-EXT*, an **optimal solution** is the graph externality $\mathsf{Ext}_{\pi^*}(G)$, yielded by the optimal allocation π^*.*

Having defined the *optimal allocation* and the *optimal solution*, we can now quantify the quality of an approximate solution for OPT-EXT, by calculating, for a specific allocation π, and therefore for a specific $\mathsf{Ext}_\pi(G)$, what percentage of the optimal solution it covers. The definition follows.

Definition 4.3.6. *For an instance* (G, O, ν) *of* OPT-EXT*, an allocation* π *is* ρ***-approximate*** *if* $\mathsf{Ext}_\pi(G) \geq \rho\mathsf{Ext}_{\pi^*}(G)$.

We can now measure the quality of the approximation algorithms we develop, by stating that a ρ-approximation algorithm for OPT-EXT provides a ρ-approximate solution or better in polynomial time, for any instance of OPT-EXT. The definition follows.

Definition 4.3.7. *For an instance* (G, O, ν) *of* OPT-EXT*, a* ρ***-approximation algorithm*** *is an algorithm that produces a* ρ*-approximate solution in polynomial time.*

Chapter 5

The OPT-EXT Problem with Two Valuations

5.1 Description and Motivation

In this section, we analyze the case where there are only two possible valuations of the objects, namely 0 or 1. It has already been shown in Proposition 1 that this case is **NP**-hard. From now on, to distinguish this problem from OPT-EXT, we will refer to the problem with valuations of only 0 and 1 as OPT-EXT(0,1) and we will refer to the OPT-EXT problem analyzed in Section 2 as the *general case* or simply OPT-EXT.

This problem draws significant attention, since it is intuitive to think about it as having two classes of objects, the "popular" ones (with valuation 1) and the "unpopular" ones (with valuation 0). This problem could also be thought as a generalization of instances where there are objects with a very high valuation and objects with a very low valuation. Note that it is important for the differences among the valuation of "same-class" objects to be significantly smaller than any difference between an object of a "low" valuation and an object of a "high" valuation for the generalization to be correct.

As a motivation for OPT-EXT(0,1), we can think again of the web page motivation example presented for the general case, but in a simplified version where objects are either *advertisements* or *posts*. We assume that posts are the ones that will draw attention from the reader, while advertisements would not draw attention by themselves, but an advertisement close to a post might draw attention from readers viewing the post.

Additionally, useful examples for OPT-EXT(0,1) can be derived from agronomy. There are certain species of plants, such as actinidias, that are male or female. The fruits (in this case, the kiwis) grow in a special case of fertilization: they grow in female trees, but only if a male tree is located nearby. This is a useful case for OPT-EXT(0,1), since it could describe a case where there are fixed locations for planting trees, and the goal is to maximize the number of female fertilized trees, with objects values 0 and 1 being female and male trees, respectively.

An instance of OPT-EXT(0,1) is quite similar to an instance of OPT-EXT, with a graph G of n vertices, k objects valued 1 and z objects valued 0. Therefore, there are m objects, with $m = k + z \leq n$. Before defining the instance of OPT-EXT(0,1), we will show that the instance we described is equivalent to assuming $k + z = n$ for the approximation of OPT-EXT(0,1).

That is, while the result can be different, the approximation algorithm process is the same as in the case where $k + z = n$ stands, by adding more objects with a valuation of 0.

Proposition 2. *Regarding the approximation of* OPT-EXT(0,1)*, we can always assume that the number of objects $(k + z)$ is equal to the number of vertices (n). If this is not the case, we can complete the instance with $n - k - z$ objects with a valuation of* 0*.*

Before moving to the proof, we will provide the definitions of a *useless zero* and a *useful zero*. Generally, a useless zero is an object with a valuation of 0 that has externality 0. The definitions follow.

Definition 5.1.1. *A* ***useless zero*** *for* OPT-EXT(0,1) *is an object o with a valuation $\nu(o) = 0$, which, under an allocation π, is allocated to a vertex v for which $ext_\pi(v) = 0$ stands.*

Definition 5.1.2. *A* ***useful zero*** *for* OPT-EXT(0,1) *is an object o with a valuation $\nu(o) = 0$, which, under an allocation π, is allocated to a vertex v for which $ext_\pi(v) = 1$ stands.*

We can now proceed to the proof of Proposition 2.

Proof. Let us consider an instance $\mathcal{I}_1$ of OPT-EXT(0,1), on a graph $G = (V, E)$, having k objects with a valuation of 1 and z_1 objects with a valuation of 0, so that $k + z_1 < n$ stands. This means that there will be some vertices without an object, formally there will be $n-k-z_1$ vertices with $\perp$ as an object.

Now consider another instance $\mathcal{I}_2$ of OPT-EXT(0,1), on the same graph $G = (V, E)$, having again k objects with a valuation of 1 and z_2 objects with a valuation of 0, so that $k + z_2 = n$ stands. Therefore, instance $\mathcal{I}_2$ will contain $\delta := z_2 - z_1 > 0$ more objects with a valuation of 0 than instance $\mathcal{I}_1$.

Consider the optimal allocations for $\mathcal{I}_1$ and $\mathcal{I}_2$ to be π_1^* and π_2^*, respectively. Let $\hat{\pi}$ be a ρ-approximate solution for OPT-EXT(0,1) on instance $\mathcal{I}_2$, for any $\rho \in (0, 1]$. According to Definition 4.3.6, we have that

$$\mathsf{Ext}_{\hat{\pi}}(G) \geq \rho \mathsf{Ext}_{\pi_2^*}(G). \tag{5.1}$$

There are two possible cases.

Case 1 — $\hat{\pi}$ contains at most δ useless zeros: In case $\hat{\pi}$ contains at most δ useless zeros, we can remove δ objects with a valuation of 0, including all the useless zeros. This means that the zeros that will remain after the removal will all be useful zeros. Formally, let us create a new allocation $\hat{\pi}'$, by removing δ items from $\hat{\pi}$, firstly the useless zeros (all, if possible) and, if δ items have not yet been deleted, remove zeros until δ items have been removed. The rest of the items will be allocated to the same vertices as in $\hat{\pi}$.

We observe that the objects in $\hat{\pi}'$ are the objects of $\mathcal{I}_1$, therefore $\hat{\pi}'$ is *feasible* for $\mathcal{I}_1$. We also see that every zero in the allocation $\hat{\pi}'$ is a useful zero, which means that $\mathsf{Ext}_{\hat{\pi}'}(G)$ for $\mathcal{I}_1$ is

maximum, therefore $\hat{\pi}'$ is an optimal allocation and yields an optimal solution for $\mathcal{I}_1$. Since $\hat{\pi}'$ is an optimal allocation for $\mathcal{I}_1$, it is also ρ-approximate, therefore in this case we have a ρ-approximate solution for $\mathcal{I}_1$. Formally, we get

$$\mathsf{Ext}_{\hat{\pi}'}(G) = \mathsf{Ext}_{\pi_1^*}(G) \geq \rho\mathsf{Ext}_{\pi_1^*}(G) \tag{5.2}$$

which concludes the proof for Case 1.

Case 2 — $\hat{\pi}$ contains more than δ useless zeros: In case the number of useless zeros in $\hat{\pi}$ is strictly larger than δ, we remove exactly δ useless zeros from $\hat{\pi}$. This way, we get an allocation $\hat{\pi}'$ that is feasible for $\mathcal{I}_1$ (since the objects in $\hat{\pi}'$ are the ones in $\mathcal{I}_1$) and also we have that the useful zeros in $\hat{\pi}$ are the same as $\hat{\pi}'$, because no useful zeros were deleted. Formally, in this case we have

$$\mathsf{Ext}_{\hat{\pi}'}(G) = \mathsf{Ext}_{\hat{\pi}}(G). \tag{5.3}$$

What remains to be shown is that $\mathsf{Ext}_{\hat{\pi}'}(G) \geq \rho\mathsf{Ext}_{\pi_1^*}(G)$. To prove this, let us construct a feasible allocation π for $\mathcal{I}_2$ based on the optimal allocation for $\mathcal{I}_1$, which is π_1^*. Specifically, we copy every item allocated to a vertex from π_1^* to π, while for any vertex v with $\pi_1^*(v) = \perp$, we allocate $\pi(v) = 0$. Formally, for every $v \in V$:

- if $\pi_1^*(v) \neq \perp$, $\pi(v) = 0$.
- if $\pi_1^*(v) = \perp$, $\pi(v) = \pi_1^*(v)$.

This way, all the useful zeros in π_1^* are also useful zeros in π. Since only useful zeros have externality, we get that the graph externality for instance $\mathcal{I}_2$ under allocation π is at least as high as the graph externality for instance $\mathcal{I}_1$ under allocation π_1^*. We also know that, for instance $\mathcal{I}_2$, the graph externality in π is lower or equal to the graph externality in π_2^*, since allocation π_2^* is an optimal allocation for $\mathcal{I}_2$. Formally, we get that

$$\mathsf{Ext}_{\pi_2^*}(G) \geq \mathsf{Ext}_{\hat{\pi}}(G) \geq \mathsf{Ext}_{\pi_1^*}(G). \tag{5.4}$$

Multiplying both sides of inequality 5.4 by ρ, we get

$$\rho\mathsf{Ext}_{\pi_2^*}(G) \geq \rho\mathsf{Ext}_{\hat{\pi}}(G) \geq \rho\mathsf{Ext}_{\pi_1^*}(G). \tag{5.5}$$

Combining inequalities 5.1, 5.5 and equality 5.3, we get that

$$\mathsf{Ext}_{\hat{\pi}'}(G) \geq \rho\mathsf{Ext}_{\pi_1^*}(G) \tag{5.6}$$

which concludes the proof for Case 2.

By concluding the proof for both cases, we get that if we have a ρ-approximate solution for $\mathcal{I}_2$, we can always get a ρ-approximate solution for $\mathcal{I}_1$, regardless of the number of useless zeros in $\mathcal{I}_2$. Therefore, the proof is concluded for Proposition 2.

■

From the proof, we get that it is sufficient to study cases with $k + z = n$ to calculate approximation ratios for cases with $k + z \leq n$. Because of that, for the rest of this section we will assume that $k + z = n$ stands, or, in other words, that the objects are exactly as many as the vertices of every instance of OPT-EXT(0,1).

5.2 Constant Approximation Algorithm for OPT-EXT(0,1)

In this subsection, an algorithm for OPT-EXT(0,1) with an approximation ratio of $(e-1)/(1+e) \approx 0.46$ is presented. Initially we will present the algorithm, then we will proceed in proving its validity and its approximation ratio.

Let us initially present an auxiliary covering problem in a graph, called AUX. Given an undirected graph G with n vertices, and two integers $b \in [0, n]$ and $r \in [0, n]$ so that $b+r \leq n$. Each vertex in the graph can receive one of three colors: blue, red or white. Additionally, at most b vertices can be blue, at most r vertices can be red, and a vertex can be red only if it has a blue neighbor. If the vertex is not blue and does not have a blue neighbor, it is white. We call a vertex *covered* if it is colored blue or red. The objective is to color every vertex of G in blue, red, or white, so as the number of *covered* vertices is maximum. The formal definition of AUX follows.

Definition 5.2.1. AUX*: Given a graph $G = (V, E)$ and two integers b, r so that $b + r \leq n$, separate the vertices in three vertex sets, B, R, W, so that:*

1. $|B| \leq b$
2. $|R| \leq r$
3. $\forall v \in R$, $\exists w \in B$ *s.t.* $w \in \mathcal{N}(v)$

and the value $|B| + |R|$ is maximum.

We can see that in order to find an optimal solution for AUX, we have to place the blue vertices in such a way that the red vertices are as many as possible. Firstly, we will prove the following helpful proposition.

Proposition 3. *Every optimal solution for* AUX *with less than b blue vertices can be transformed into an optimal solution for* AUX *with exactly b blue vertices.*

Proof. Consider the sets B^*, R^*, W^* to be the sets of an optimal solution of an instance of AUX on a graph $G = (V, E)$, with $B^* < b$. Assume that there is at least one white vertex v. By coloring v blue, we will still have $|B^* \cup v| \leq b$, but we will have $|B^* \cup v| + |R^*| > |B^*| + |R^*|$, which means the new solution will be feasible but better than the optimal one. Since this cannot be true, it follows that $W^* \equiv \emptyset$, therefore there are no white vertices and $|B^*| + |R^*| = |V|$. Therefore, we can change the color for $b - |B^*|$ red vertices to blue, with all the vertices still being covered. Thus, every optimal solution for AUX has or can be transformed to have exactly b blue vertices. ■

Therefore, we can assume without loss of generality that every optimal solution for AUX has exactly b blue vertices.

We can observe that an instance for AUX can be derived from an instance of OPT-EXT(0,1) in the following way: $G = (V, E)$ is identical, $b = k$ and $r = z$. Therefore, a solution for OPT-EXT(0,1) can be derived from a solution to AUX, by allocating objects with valuation 1 to blue vertices and objects with valuation 0 to red and white vertices. From Proposition 3, we can ensure that the number of blue in the solution of AUX will be indeed equal to k, therefore the number of vertices with a valuation of 1 in the respective instance of OPT-EXT(0,1) will be equal to k. From Proposition 2, we can also consider without loss of generality that $b + r = n$, since for the respective instance of OPT-EXT(0,1) we consider that the number of objects is equal to the number of vertices. Moreover, note that red vertices are *useful zeros* and white vertices are *useless zeros*, as stated in definitions 5.1.2 and 5.1.1, respectively, since red vertices are neighboring a vertex with a valuation of 1, while white vertices are not. This means that the total graph externality for an instance of OPT-EXT(0,1) is equal to the number of the red vertices of the solution for an instance of AUX.

Using Algorithm 4, we produce an approximate solution $sol(b)$ to AUX on a graph $G = (V, E)$ with $b = k$ and $r = z$.

Algorithm 4

Input: b, r, G of order n

Output: A feasible solution to AUX

1: Every vertex of G is white.
2: $sol(0) \leftarrow \emptyset$
3: **for** $t = 1$ to b **do**
4: Color in blue a vertex v that is not already blue, and if possible, color in red some white neighbors of v. Do this so as to maximize the number of newly covered vertices, under the constraint that the total number of red vertices is at most r.
5: $sol(t) \leftarrow sol(t-1) \cup \{\text{the newly covered vertices}\}$
6: **end for**
7: **return** $sol(b)$

Theorem 5.2.2. *There exists a* $(e-1)/(1+e)$ *approximation algorithm for* OPT-EXT(0,1).

Proof. Again, consider S^* to be the set of covered vertices in an optimal coloring for AUX. We will initially describe a way to partition S^* to b sets.

Take the blue vertex v_1 that has the largest number of red neighbors (ties can be broken arbitrarily). Consider the set of the red vertices in $\mathcal{N}(v)$ along with v_1 to be S_1^*. Now, take the next blue vertex v_2 having the largest number of red neighbors which are not included in a previous set. Include these first-appearing next neighbors and the vertex v_2 in another set denoted as S_2^*. Repeat this process until there are b sets, namely $S_1^*, S_2^*, ..., S_b^*$. Note that some sets can have a size of 1, with only the respective blue vertex being in the set. Moreover, note that the sets $S_1^*, S_2^*, ..., S_b^*$ are *disjoint*, meaning that every element can only be in at most one of the sets. Eventually, each one of these b disjoint sets has exactly one blue vertex. Consider $t^* \leq b$ to be the largest index so that each set in the collection $S_1^*, S_2^*, ..., S_{t^*}^*$ has at least one red vertex. Note that either $t^* = b$ (in which case, every set will contain at least one red vertex), or none of the sets $S_{t^*+1}^*, S_{t^*+2}^*, ..., S_b^*$ will contain red vertices.

We consider allocations π_1 and π^* to be the allocations derived from $sol(b)$ (the approximate solution for AUX which is the result of Algorithm 4) and $S^* := |B^*|+|R^*|$ (the optimal solution for AUX), respectively. For allocation π^* we know that the graph externality derived by it is the size of the sets $S_1^*, S_2^*, ..., S_{t^*}^*$ excluding the blue vertices, since every other vertices (the red ones) in these sets derive externality, while there are no vertices in the sets $S_1^*, S_2^*, ..., S_{t^*}^*$ that derive externality. Formally. we have

$$\mathsf{Ext}_{\pi^*}(G) = |\bigcup_{i=1}^{t^*} S_i^*| - t^*. \tag{5.7}$$

For the allocation π_1, we know that its externality is at least as high as the coverage yielded from Algorithm 4 implemented on the vertices of the sets $S_1^*, S_2^*, ..., S_{t^*}^*$, if we subtract the blue vertices. Formally, we have

$$\mathsf{Ext}_{\pi_1}(G) \geq |sol(t^*)| - t^*. \tag{5.8}$$

Using a similar method to the one described in [30, Lemma 3.14], we can prove the following inequality regarding $|sol(t^*)|$:

$$|sol(t^*)| \geq (1 - 1/e)|\bigcup_{i=1}^{t^*} S_i^*|. \tag{5.9}$$

Combining inequalities 5.8 and 5.9, we have

$$\mathsf{Ext}_{\pi_1}(G) \geq (1 - 1/e)|\bigcup_{i=1}^{t^*} S_i^*| - t^*. \tag{5.10}$$

For any $\alpha \in (0, 1]$ we can transform inequality 5.10 to

$$\mathsf{Ext}_{\pi_1}(G) \geq (1 - 1/e)|\bigcup_{i=1}^{t^*} S_i^*| - \alpha t^* - (1-\alpha)t^*. \tag{5.11}$$

Let us consider the number $\delta := (3-e)/(e-1)$. We can now consider two cases:

Case 1 — $t^* \leq |\bigcup_{i=1}^{t^*} S_i^*|/(3+\delta)$. In this case, inequality 5.11 becomes

$$\mathsf{Ext}_{\pi_1}(G) \geq \left(1 - \frac{1}{e} - \frac{\alpha}{3+\delta}\right)|\bigcup_{i=1}^{t^*} S_i^*| - (1-\alpha)\,t^*. \tag{5.12}$$

Since we can pick any value of $\alpha \in (0,1]$, using $\alpha = (3+\delta)/e(2+\delta)$ we get from inequality 5.12 that

$$\mathsf{Ext}_{\pi_1}(G) \geq \left(1 - \frac{3+\delta}{e(2+\delta)}\right)\left(|\bigcup_{i=1}^{t^*} S_i^*| - t^*\right) = \frac{e-1}{1+e}\left(\mathsf{Ext}_{\pi^*}(G)\right) \tag{5.13}$$

Therefore, in this case, we have a $(e-1)/(1+e)$-approximate solution for OPT-EXT(0,1).

Case 2 — $t^* > |\bigcup_{i=1}^{t^*} S_i^*|/(3+\delta)$. In this case, the number of red vertices in π^* (which is $|\bigcup_{i=1}^{t^*} S_i^*| - t^*$ can be shown to be

$$|\bigcup_{i=1}^{t^*} S_i^*| - t^* < (2+\delta)t^*. \tag{5.14}$$

Since the number of red vertices is equal to the total graph externality (from equation 5.7) we have

$$\mathsf{Ext}_{\pi^*}(G) < (2+\delta)t^*. \tag{5.15}$$

Consider now a second allocation π_2 obtained in the following steps. Firstly, construct a maximum matching M of G (the definition of a maximum matching is included in section 2.1). Out of the edges of M, choose $min(k, z, |M|)$ of them arbitrarily. For each edge, allocate an object of value 1 and an object of value 0 to its adjacent vertices arbitrarily. Then, allocate the remaining objects arbitrarily to vertices, obtaining the allocation π_2.

In allocation π_2, we can observe that at least one red vertex is attached to each of the t^* blue vertices. Therefore, allocation π_2 yields a solution of at least t^*. But from inequality 5.15 we have that

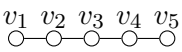
$$t^* > \frac{\mathsf{Ext}_{\pi^*}(G)}{2+\delta}. \tag{5.16}$$

This means that the solution to OPT-EXT(0,1) yielded from π_2 is a $1/(2+\delta)$-approximate solution, with $1/(2+\delta) = (e-1)/(1+e)$. Therefore we obtain a $(e-1)/(1+e)$-approximate solution in this case as well, which concludes the proof. ■

Therefore, by using the allocation out of $\{\pi_1, \pi_2\}$ depending on which case we are at, we obtain a solution that has a value at least $(e-1)/(1+e) \approx 0.46$ the value of the optimal solution.

Note that this limit is not *tight*, meaning that it is not yet proven that the approximation ratio is indeed $(e-1)/(1+e)$ and not higher. We will now provide an example where the algorithm performs 2/3 of the optimal solution. This is an upper bound for the approximation ratio of the algorithm, because it proves that the approximation ratio cannot be higher than 2/3.

Consider the following graph with 2 objects of value 1 and 3 objects with value 0.



The approximation algorithm can place the objects of value 1 on v_3 and v_5; the resulting externality is 2. The optimal solution, of externality 3, places the objects of value 1 on vertices v_2 and v_4. Thus, the approximation ratio of the algorithm is at most 2/3.

Still, there is a gap between $(e-1)/(1+e) \approx 0.46$ and 2/3, which triggers the question of determining the exact approximability of OPT-EXT(0,1).

For certain *coverage problems*, there exist greedy algorithms whose objective function is *monotone submodular*. An example for that is [41] for a $(1-(1-1/k)^k$-approximation algorithm, where $(1-(1-1/k)^k \geq 1-1/e$. A monotone function can be thought as an increasing or a decreasing function over a finite ordered set. A submodular function can be thought as a function whose the difference of the incremental value brought by an extra element added to the set decreases as the size of the set increases. The formal definitions for *monotone* and *submodular* functions follow.

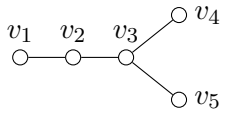
Definition 5.2.3. *A function Φ is **monotone** over a finite set Ω, if for every X, Y with $X \subseteq Y \subseteq \Omega$, $\Phi(X) \leq \Phi(Y)$ stands.*

Definition 5.2.4. *A function Φ is **submodular** over a finite set Ω, if for every X, Y with $X \subseteq Y \subseteq \Omega$, for any $u \in \Omega \setminus Y$, $\Phi(X \cup \{u\}) - \Phi(X) \geq \Phi(Y \cup \{u\}) - \Phi(Y)$ stands.*

Despite using results for coverage problems in the proof process for Theorem 5.2.2, we are going to show why OPT-EXT(0,1) does not appear to reduce to the maximization of a monotone submodular function.

Given an instance (G, O, ν) of OPT-EXT(0,1), let us define a function $f : 2^V \to \mathbb{N}$ as follows. For any $S \subseteq V$ so that $|S| \leq k$, $f(S)$ is the minimum between z and the number of vertices in $V \setminus S$ which have a neighbor in S. In other words, if we allocate k objects valued 1 to vertices of G and we allocate objects valued 0 to at most z of the neighbors of the vertices of

S, we get that $f(S)$ is the externality of the allocation. In the following instance, we observe that f is neither monotone nor submodular.



3 objects with value 1

2 objects with value 0

Let $X = \{v_2\}$, $Y = \{v_2, v_3\}$ and $u = v_1$. Therefore, we get $f(\{v_1, v_2\}) - f(\{v_2\}) = 1 - 2 < 2 - 2 = f(\{v_1, v_2, v_3\}) - f(\{v_2, v_3\})$. The violation of monotonicity exists since $f(\{v_2\}) = 2 > f(\varnothing) = 0$, but simultaneously $f(\{v_1, v_2\}) = 1 < f(\{v_2\}) = 2$. The violation of submodularity exists since $f(\{v_1, v_2\}) - f(\{v_2\}) < f(\{v_1, v_2, v_3\}) - f(\{v_2, v_3\})$.

While OPT-EXT(0,1) resembles many covering problems, it is important to highlight a key difference between OPT-EXT(0,1) and other covering problems. In MAX COVERAGE there is a distinction between elements to be covered and sets that cover them. In DOMINATING SET, every vertex dominates itself and its neighbors, yielding always "positive" value towards the percentage of the graph vertices that are dominated. In contrast, a vertex in OPT-EXT(0,1 can either exert externality or derive, but not both.

5.3 Relation with Partial Domination

In this section, we are presenting exact algorithms (and not approximation ones) for OPT-EXT(0,1), built upon exact algorithms for PARTIAL DOMINATING SET (PDS for short).

In order to define the algorithms and describe the building process, we need to define PDS as a problem. Like DOMINATING SET, the concept of *dominating* vertices stands, but here we do not need to cover the whole set of vertices V, but we need to cover t vertices using as few vertices as possible. The definition for PDS follows.

Definition 5.3.1. PARTIAL DOMINATING SET (PDS)*: Given a graph $G = (V, E)$ and an integer $t \geq 0$, calculate a set of vertices $S \subseteq V$ so that $\left|\bigcup_{v \in S} \mathcal{N}[v]\right| \geq t$ and $|S|$ is minimum.*

Let D_t denote an optimal set of vertices that dominates t vertices of G, that is, let D_t be an optimal solution for PDS. Demaine *et al.* [13] have shown that D_t can be computed in time $3^{1.5\mathsf{tw}} n^{\mathcal{O}(1)}$, where tw and n are the treewidth and the number of vertices in G, respectively. Later, Fomin *et al.* [15] proposed a subexponential algorithm for PDS in apex-minor-free graphs (this class comprises planar graphs) with a running time of $2^{\mathcal{O}(\sqrt{|D_t|})} n^{\mathcal{O}(1)}$.

Now, we will propose an exact algorithm for OPT-EXT(0,1), based on some observations from PDS.

Theorem 5.3.2. *Every algorithm that solves* PDS *in time $T(n)$ yields a $nT(n)$ time exact algorithm for* OPT-EXT(0,1).

Proof. Let D_t denote an optimal solution for PDS, that is, a minimum size set of vertices dominating at least t vertices of G. Let us suppose without loss of generality that D_t can be computed in $T(n)$ time. Perform the the computation of D_t exactly n times, for t from 1 to n inclusive. For each D_t, consider F_t to be the set of vertices in $V \setminus D_t$ that are dominated by at least one vertex of D_t. Consider t^* to be the index at which F_t is maximized under the constraint $|D_t| \leq k$.

Consider an allocation π for OPT-EXT(0,1) for the graph G and a set of objects O with their valuations $\mathcal{N}$ as follows. Allocate objects valued 1 to every vertex of D_{t^*}. It is clear that since $|D_{t^*}| \leq k$, there will be at least $|D_{t^*}|$ objects valued 1 to be allocated to these vertices. Now, allocate objects valued 0 to every vertex of F_{t^*}, until all objects valued 0 are allocated or F_{t^*} has no vertices remaining. If needed, complete the allocation arbitrarily, allocating the remaining objects to the remaining vertices.

We need to prove the correctness of the algorithm for the following two cases.

Case 1 — All objects valued 0 are allocated to vertices in F_{t^*}. If all objects valued 0 are allocated to vertices in F_{t^*}, it follows that π is an optimal allocation, since every object in O derives the maximum possible externality, since it has a neighbor of valuation 1, so in this case the algorithm is correct.

Case 2 — At least one object valued 0 does not derive externality. If at least one object valued 0 does not get externality, then suppose that π is not an optimal allocation. This means that there is another allocation $\tilde{\pi}$ with a higher graph externality. Let $D_{\tilde{t}}$ be the set of vertices with an object valued 1 in $\tilde{\pi}$ and let $F_{\tilde{t}}$ be the set of vertices with an object valued 0 and deriving externality 1 in $\tilde{\pi}$. Since $|D_{\tilde{t}}| \leq k$, this means that there exists a solution for PDS for which $|D_{\tilde{t}}| \leq k$ and $|F_{\tilde{t}}| > |F_{t^*}|$. This would mean, however, that the solution for PDS yielding the set F_{t^*} is not an optimal solution, which is a contradiction. Therefore, in this case the algorithm is correct as well, which concludes the proof. ■

In section 2.1, we explained the analytical definitions of *treewidth* and *apex-minor-free* graphs. According to theorem 5.3.1 and based on the work of Demaine *et al.* and Fomin *et al.*, we get two exact algorithms for OPT-EXT, described in corollaries 5.3.2.1 and 5.3.2.2.

Corollary 5.3.2.1. *There exists an exact algorithm for* OPT-EXT *for graphs with a treewidth of at most 10 with a running time of* $n \cdot 3^{1.5\mathsf{tw}} n^{\mathcal{O}(1)}$.

Corollary 5.3.2.2. *There exists an exact algorithm for* OPT-EXT *for apex-minor-free graphs with a running time of* $n \cdot 2^{\mathcal{O}(\sqrt{|D_t|})} n^{\mathcal{O}(1)}$.

Chapter 6

The OPT-EXT Problem with General Valuations

In this section, we analyze OPT-EXT without the objects being restricted to values 0 and 1. Proposition 2 is not valid here, so we do not assume that $|O| = |V|$.

Given an allocation π and a graph $G = (V, E)$, we associate a digraph $\mathcal{D}$ with vertex set V and arc set A, denoted as $\mathcal{D} = (V, A)$. There is an arc $(v, w) \in A$ if v derives externality from w. Since a vertex can derive externality from at most one neighbor, the *out-degree* of every vertex in $\mathcal{D}$ is at most 1. Considering Example 4.2 again, the digraph associated with the graph in the example has 3 arcs: (v_3, v_1), (v_4, v_5), and (v_5, v_1).

We observe that if a vertex v derives externality from w, the amount by which the arc (v, w) contributes to $\mathsf{Ext}_\pi(G)$ is $\nu(\pi(w)) - \nu(\pi(v))$. Summing all the values for all the arcs, we see that we can formulate the graph externality as a dot product as follows:

$$\mathsf{Ext}_\pi(G) := \sum_{v \in V \text{ s.t. } \pi(v) \neq \perp} h(v) \cdot \nu(\pi(v)) \tag{6.1}$$

where $h(v)$ is defined as the in-degree of v in $\mathcal{D}$ minus the out-degree of v in $\mathcal{D}$ and $\nu(\pi(v))$ is the valuation of the object allocated to vertex v. In other words, $h(v)$ is the number of vertices deriving externality from v, minus 1 if v derives externality from one of its neighbors.

6.1 Graphs with Maximum Degree 2

When every vertex in a graph $G = (V, E)$ has a maximum degree of 2, we observe that the graph is a collection of connected components, with each connected component being a **path** or a **cycle**. We will prove that in this case, we can get an exact polynomial time algorithm for OPT-EXT. However, when the maximum degree is 3 or more, by Proposition 1 we know that OPT-EXT is **NP**-hard.

Theorem 6.1.1. OPT-EXT *can be solved in polynomial time for an instance* (G, O, ν) *when* G *has a maximum degree of at most* 2*.*

Proof. We are going to construct an allocation to prove Theorem 6.1.1. The allocation is

going to be constructed in two stages.

In the first stage, we partially cover G with a collection $P_1, P_2, ..., P_z$ of disjoint paths. Each path P_ℓ has a length of at most 2 (the length of the path is the number of edges it includes, or, equivalently, the number of vertices it includes minus 1). The collection of paths, in total, covers exactly $|O|$ vertices. We use algorithm 5 for the first stage of the algorithm.

In the second stage, we construct the final optimal allocation π using algorithm 6.

First stage: Consider the digraph D^* associated with G and π^*, where π^* is an optimal allocation to OPT-EXT. The digraph is constructed in the way it is described in the start of section 6.

We are going to describe an operation called *Reversal*. Suppose D^* contains a directed path of length k, where k is at least 2. Let the path be $((v_i, v_{i+1}), ..., (v_{i+k-1}, v_{i+k}))$. Modify π^* by reversing the allocation for the vertices of the path, excluding the last vertex, that is, excluding vertex v_{i+k}. This means that vertex v_{i+k-1} will swap objects with vertex v_1, vertex v_{i+k-2} will swap objects with v_2 and so on, until no more swaps are available in the path. We observe that the total externality throughout the path remains the same, since before the reversal, the total externality in the path was

$$(\nu(\pi^*(v_{i+k})) - \nu(\pi^*(v_{i+k-1}))) + (\nu(\pi^*(v_{i+k-1})) - \nu(\pi^*(v_{i+k-2}))) + ... + (\nu(\pi^*(v_2)) - \nu(\pi^*(v_1))) \tag{6.2}$$

while after the reversal, the total externality is

$$\nu(\pi^*(v_{i+k})) - \nu(\pi^*(v_1)). \tag{6.3}$$

It is clear that the value in (6.2) is the same as the value in (6.3). We observe, however, that the number of arcs in the path has been reduced from k to 1.

Repeat *Reversal* on the allocation π^* until it is no longer possible. It is clear that this is a finite process, since every time a *Reversal* is performed, the total number of arcs in the digraph is strictly smaller than the total number of arcs before the *Reversal*. After the repetition of *Reversal* until it is no longer possible to perform it again, we observe that $\mathsf{Ext}_{\pi^*}(G)$ is the same, therefore π^* remains an optimal allocation. Since every vertex in G has a degree of at most 2, we observe that the digraph D^* associated with π^* has connected components with up to 3 vertices each. There are now only 4 possible scenarios for every vertex in the digraph D^*:

- In-degree 2 and out-degree 0
- In-degree 1 and out-degree 0
- In-degree 0 and out-degree 0

- In-degree 0 and out-degree 1

There scenario of in-degree 1 and out-degree 1 for a vertex is no longer possible, since that would imply the existence of a path of length at least 2 in the digraph D^*, which would mean that *Reversal* could be repeated at least once more. Therefore, using the notation of $h(v)$ from equation 6.1, we get that for every $v \in V$, we have $h(v) \in \{2, 1, 0, -1\}$.

We can see if there are two vertices v and w so that $\nu(\pi(v)) > \nu(\pi(w))$ and $h(v) < h(w)$, we can swap their objects and strictly increase $\mathsf{Ext}_{\pi^*}(G)$. This can be explained, as the object with higher valuation will either give more externality to its new neighboring vertices after the swap, or/and the object with the smaller valuation will derive more externality from its new neighbor with the highest valuation after the swap. Therefore, since π^* is an optimal allocation, it follows that if $h(v) < h(w)$ (for any $v, w \in V$), $\nu(\pi(v)) \leq \nu(\pi(w))$ must hold. In other words, under the allocation π^*, the objects ordered by non-increasing valuation are placed on the vertices of G ordered by non-increasing h-value. Using again the dot product formula of equation 6.1, we can see that under the optimal allocation π^*, the total externality $\mathsf{Ext}_{\pi}(G)$ is a dot product $\vec{\nu} \cdot \vec{x}$, where $\vec{\nu}$ (respectively, $\vec{x}$) consists of $\{\nu(o) : o \in O\}$ (respectively, $\{h(v) : v \in V\}$) sorted in non-increasing order.

Now, consider the output π of Algorithm 6 and let D^* be its associated digraph. As far as the optimal solution is concerned, it consists of connected components of at most 3 vertices and the objects ordered by non-increasing valuation are placed on the vertices of G ordered by non-increasing h-value. Therefore, $\mathsf{Ext}_{\pi}(G)$ is also a dot product $\vec{\nu} \cdot \vec{y}$, where $\vec{\nu}$ (respectively, $\vec{y}$) consist of $\{\nu(o) : o \in O\}$ (respectively, $\{h(v) : v \in V\}$) sorted in non-increasing order.

The possible difference between $\vec{x}$ and $\vec{y}$ comes from the h-values. By definition, the sum of the coordinates of both $\vec{x}$ and $\vec{y}$ is 0. Since $\vec{y}$ is, by construction, lexicographically larger than $\vec{x}$, it follows that $\mathsf{Ext}_{\pi}(G) = \vec{\nu} \cdot \vec{y} \geq \vec{\nu} \cdot \vec{x} = \mathsf{Ext}_{\pi^*}(G)$. In other words, algorithms 5 and 6 solve OPT-EXT optimally. ■

The analytical step-by-step description of algorithms 5 and 6 used in the proof follows.

Algorithm 5 Algorithm for the first stage of the proof of Theorem 6.1.1

Input: $|O|$ and G which has maximum degree 2

Output: A set of disjoint sub-paths of G, each of length at most 2, which spans exactly $|O|$ vertices of G

Remove an arbitrary edge of each cycle of G so that G becomes a collection of paths
$spn \leftarrow 0$ {spn is the number of vertices spanned so far}
$z \leftarrow 0$ {z is the number of sub-paths built so far}
while $spn < |O|$ **do**
    $z \leftarrow z + 1$
    Let s be the minimum between 3, 1+the length of the longest path of G, and $|O| - spn$
    Choose a sub-path of G on s vertices and call it P_z (P_z must contain a vertex whose degree in G is 1)
    $G \leftarrow G \setminus P_z$
    $spn \leftarrow spn - s$
end while
return $P_1, \ldots, P_z$

Algorithm 6 Algorithm for the second stage of the proof of theorem 6.1.1

Input: G, O and a collection $P_1, \ldots, P_z$ of paths of length at most 2

Output: An allocation π

In the collection, each path P_ℓ of length 2 consists of 3 contiguous vertices whose center is denoted by c_ℓ. Each path P_ℓ of length 1 consists of 2 contiguous vertices; choose one of them arbitrarily to be the center c_ℓ. Each path of length 0 consists of a single vertex which is the center
π is initially empty
for $\ell = 1$ to z **do**
    Let o^* be the object with largest valuation in O
    $\pi(c_\ell) \leftarrow o^*$
    $O \leftarrow O \setminus \{o^*\}$
end for
Complete π by placing arbitrarily the remaining objects on the free vertices (i.e. the non-centers) of $P_1, \ldots, P_\ell$
return π

6.2 Caterpillar Trees

There are quite a lot **NP**-hard problems that have polynomial time solutions on certain graphs such as trees. We remind here that a **tree** is a graph that does not contain cycles. For instance, DOMINATING SET has a polynomial time solution for trees. It is meaningful to consider tree graphs for instances of OPT-EXT, since they represent a hierarchy, while generally, situations that are represented as tree graphs in real life are quite common.

This section contains certain results for a specific case of tree, the **caterpillar tree**. Let us describe the procedure of constructing a caterpillar tree of n vertices. Initially, consider a path of $k \leq n$ vertices. We call this path the **backbone**, with the name *backbone* being selected as the path can be thought as the backbone of a caterpillar. The remainining $n - k$ vertices are called **pendant** vertices, and each one of them has degree 1, always neighboring a vertex from the k backbone vertices. Thus, each one of the *pendant* vertices can be thought as the legs of a caterpillar, explaining the name *caterpillar tree*.

We define a *star* to be a graph with a vertex in the center and every other vertex having a degree of 1 and neighboring the center vertex. The definition of the star follows.

Definition 6.2.1. *A graph $G = (V, E)$ is a **star** when there is a vertex $v \in V$ of degree $|V| - 1$, which is called the **center** of the star, for which $\forall w \in \{V \setminus \{v\}\}$: $deg(w) = 1$ and $\{v, w\} \in E$.*

The *degree* of a star is the degree of the center of the star, which equals the number of vertices in the star minus 1. To derive a result regarding OPT-EXT in caterpillar trees, let us first prove the following useful proposition.

Proposition 4. OPT-EXT *can be solved in polynomial time when $G = (V, E)$ is a collection of x stars.*

Proof. Let us initially order the objects of O in non-increasing order of their respective valuations and let us order the x stars in non-increasing order of their degree, getting an ordering $S_1, S_2, ..., S_x$. Consider the list of the non-allocated objects to be the list of *non-allocated objects*, denoted by L, getting an ordering $o_1, o_2, ..., o_m$ for the m objects.

Let us describe a way to create an allocation π. Consider the first star in the ordering of the stars, let. Pick the first object (i.e. the one with the highest valuation) in L, assign it to the center of S_1, then remove it from L. Start picking elements from the end of L (i.e. the ones with the lowest valuations) and assign them to the neighbors of the center of S_1, removing them upon allocation, repeating this up until either all the objects are allocated, or every vertex in S_1 has an object allocated to it. Note that, if the graph was only S_1, the solution would have been optimal, as the graph externality derived would be the maximum possible, since the object with the largest valuation is allocated to the center of S_1 and the objects with the lowest valuations are allocated to its neighbors. If more objects were remaining, we repeat this process until all objects are allocated to vertices ($|O| \leq |V|$ continues to hold).

It follows that $y \leq x$ stars from S, namely $S_1, S_2, ..., S_y$, will be the ones with a vertex allocated in their centers after the formation of π is completed. This means that o_1 is allocated to the center of S_1, o_2 is allocated to the center of S_2 and so on. Note that all x stars may not have been used, since all the objects may have been allocated to vertices before reaching star x in the aforementioned repetition. In other words, $y \leq x$, but it could also hold that $y < x$. Since we sorted the objects in non-increasing order, we know that $\nu(o_1) \geq \nu(o_2) \geq ... \geq \nu(o_y)$. We also know that since we sorted the stars in non-increasing order of degree, we have $deg(\pi^{-1}(o_1)) \geq deg(\pi^{-1}(o_2) \geq ... \geq deg(\pi^{-1}(o_y))$ (this regards the degrees of the centers of the stars). We observe that the objects $o_{y+1}, o_{y+2}, ..., o_m$ are going to be the ones deriving externality, which is optimal, since in the way described, we have as many objects as possible deriving externality, and in this case we have the objects with the lowest valuations deriving externality. Note that there is not any other way for more than $m - y$ objects to derive externality. Since the $m - y$ objects that derive externality are all of lower or equal valuation to the y objects that yield externality to them, we note that any of the y highest valuation objects could yield externality to any of the $m - y$ lowest valuation objects. Formally, we observe that the total graph externality can be formulated as

$$\mathsf{Ext}_\pi(G) = \sum_{i=1}^{y-1} \big(deg(\pi^{-1}(o_i)) \cdot \nu(o_i)\big) + f(\pi^{-1}(o_y)) \cdot \nu(o_y) - \sum_{j=y+1}^{n} \nu(o_j), \tag{6.4}$$

where $f(\pi^{-1}(o_y)$ is equal to the number of objects allocated to neighbors of the y-th star (note that since it is the last star to be filled, the objects might run out before all the neighbors of its center have objects allocated to them). Observe that the value in (6.4) remains the same, even if we randomize the placement of the last $m - y$ objects (among the vertices that are neighbors of the centers, even among different stars).

We observe that the only value that changes in (6.4) if the placement of the highest valued y objects is randomized (among the centers) is $deg(\pi^{-1}(o_i))$ for objects from 1 to $y - 1$ and $f(\pi^{-1}(o_y)$ for the y-th object. Therefore, to maximize the total graph externality, the optimal solution is to allocate o_1 to the vertex with the highest degree, then to allocate o_2 to the vertex with the next highest degree and so on. This concludes the proof that π is an optimal allocation. ■

We are now going to propose a 0.5-approximation algorithm for the case of the caterpillar tree, which relies on solving OPT-EXT on a subgraph of G which is a path. We are going to use the following Lemma to assist the proof of the ratio of the 0.5-approximation algorithm, with the proof being derived in a similar manner to the one of 6.1.1 of the OPT-EXT(0,1) case.

Lemma 6.2.2. OPT-EXT *can be solved in polynomial time on a path having strictly less than* $|O|$ *vertices.*

Proposition 5. *A* 0.5*-approximation algorithm exists for* OPT-EXT *when* G *is a caterpillar.*

Proof. Let us consider an optimal allocation π^* for the caterpillar tree. Let us denote the externality for the backbone vertices under π^* as $\mathcal{E}_b^*$ and the externality for the pendant

vertices under π^* as $\mathcal{E}_p^*$.

Consider vertices $v_1, v_2, ...v_b$ to be the *backbone* vertices and vertices $v_{b+1}, v_{b+2}, ..., v_n$ to be the *pendant* vertices. Using Lemma 6.2.2, we know that an optimal allocation π_1 for the backbone vertices can be computed in polynomial time. Respectively, using Proposition 4, an optimal allocation π_2 for the pendant vertices can be computed in polynomial time as well. Therefore, we have that

$$\mathsf{Ext}_{\pi_1}(\{v_1, v_2, ..., v_b\}) \geq \mathcal{E}_b^* \tag{6.5}$$

and

$$\mathsf{Ext}_{\pi_2}(\{v_{b+1}, v_{b+2}, ..., v_n\}) \geq \mathcal{E}_p^*. \tag{6.6}$$

But we also know that

$$\mathsf{Ext}_{\pi^*}(G) = \mathcal{E}_b^* + \mathcal{E}_p^* \tag{6.7}$$

which means that either

$$\mathsf{Ext}_{\pi_1}(G) \geq \mathsf{Ext}_{\pi_1}(\{v_1, v_2, ..., v_b\}) \geq 0.5\mathsf{Ext}_{\pi^*}(G) \tag{6.8}$$

or

$$\mathsf{Ext}_{\pi_2}(G) \geq \mathsf{Ext}_{\pi_2}(\{v_{b+1}, v_{b+2}, ..., v_n\}) \geq 0.5\mathsf{Ext}_{\pi^*}(G). \tag{6.9}$$

Therefore, selecting the best out of the allocations π_1 and π_2, we get a 0.5-approximate solution for OPT-EXT in caterpillar trees.

■

Chapter 7

Experimental Results

In this section, we present a greedy algorithm for OPT-EXT and we evaluate experimentally its performance.

Let us firstly define an upper bound on the optimal solution of OPT-EXT. This means that, since this number is an upper bound, the optimal solution does not yield a higher graph externality than this upper bound.

Let us call a trivial upper bound $T(G)$ on the optimal externality. Given an instance $G(V, E)$ and O, consider a star S_T of degree $|O| - 1$, with the object of maximum valuation in its center and all other objects on its leaves. Let the objects to be sorted in non-increasing order of valuation, i.e., $\nu(o_1) \geq \nu(o_2) \geq ... \geq \nu(o_m)$. Because of the way the leaf is formed, we have that the externality that a leaf of S_T that is assigned an object o_i derives, is equal to $\nu(o_1) - \nu(o_i)$. Letting $T(G)$ be the total externality of S_T, therefore we have

$$T(G) = \nu(o_1) \cdot (|O| - 1) - \sum_{i=2}^{m} \nu(o_i) \tag{7.1}$$

or, simply

$$T(G) = \nu(o_1) \cdot |O| - \sum_{i=1}^{m} \nu(o_i). \tag{7.2}$$

Note that $T(G)$ is a trivial upper bound, since every object yields the maximum possible externality. It is clear that there are no other instances with the same object set O that have a higher graph externality. Therefore, denoting the optimal allocation of an instance of OPT-EXT as π^*, and denoting the graph externality we get from the optimal allocation as $\mathsf{Ext}_{\pi^*}(G)$, we have that

$$\mathsf{Ext}_{\pi^*}(G) \leq T(G). \tag{7.3}$$

We are now going to define a nontrivial upper bound $U(G)$ for OPT-EXT, which is lower than $T(G)$, but still an upper bound to the optimal solution. Consider the degrees of V (where V

is the vertex set of $G = (V, E)$) sorted in non-increasing order, let them be $d_1, d_2, ..., d_n$. Let the corresponding vertices to the degrees be $v_1, v_2, ..., v_n$.

We recall that $T(G)$ was derived by considering that all the vertices were able to be the center and the leaves of one star of size $n-1$. We will prove that we can get an upper bound from a *collection* of smaller stars, each one having as many leaves as the degrees $d_1, d_2, ..., d_n$. We will prove that this upper bound yields not higher externality than $T(G)$ and not lower externality than $\mathsf{Ext}_{\pi^*}(G)$. Note that we are not going to use n stars, but we are going to use k stars, which should be enough to fit the k highest valued objects in their centers, and the rest of the objects in their leaves. Formally, consider k to be the lowest integer satisfying the inequality

$$k + \sum_{i=1}^{k} d_i \geq |O|. \tag{7.4}$$

Then, we can consider a collection of stars S_U with centers $v_1, v_2, ..., v_k$ for the stars $S_1, S_2, ..., S_k$, respectively. It follows that for every star S_i, we have

$$deg(S_i) = deg(v_i) = d_i. \tag{7.5}$$

Note that by this definition, some vertices of G may appear multiple times in S_U. In other words, it is possible (and quite frequent) for the same vertex to appear in different stars of S_U.

Let us create an allocation π of O to the vertices of S_U as follows. Let us firstly assign the objects with the largest k valuations to the centers of S_U, in a manner that o_1 is allocated to v_1, o_2 is allocated to v_2, ..., o_k is allocated to v_k. Now, allocate the remaining objects $o_{k+1}, o_{k+2}, ..., o_m$ in non-decreasing order of valuation to the leaves of the stars, starting with star S_1 and allocating the objects until its leaves all have an object before moving to the next star. Note that it is possible for some neighbors of v_k not to get any object. In fact, the only way for every neighbor of v_k to all get an object is for the equation $d_1 + d_2 + ... + d_k = |O| - k$ to stand. Additionally, note that since some vertices of G may appear multiple times in S_U, in this scenario we assume that if the same vertex $v \in V$ appears twice in stars S_i, S_j, that the vertex from S_i is a completely different vertex from the vertex in S_j, therefore two different objects can be allocated to these vertices.

Let $U(G)$ be the total externality derived from the leaves of under the aforementioned allocation. It is clear that $U(G)$ may not be feasible under the instance of graph G, since the stars of S_U were created in a manner that could have the same vertex to appear in different stars. However, we can tell that it provides an upper bound to $\mathsf{Ext}_{\pi^*}(G)$, which is usually better (i.e. closer to the optimal solution) than $T(G)$. We are going to prove this by proving the following proposition.

Proposition 6. *For any instance* (G, O) *of* OPT-EXT *and any allocation* π*, we have* $\mathsf{Ext}_{\pi}(G) \leq U(G) \leq T(G)$.

Proof. We have already shown earlier through equation 7.3 that $T(G)$ is an upper bound of the optimal solution, since all objects, apart from the one with the highest valuation, derive the maximum possible externality. The object with the highest valuation cannot derive any externality by definition, therefore $T(G)$ is an upper bound of the optimal solution and thus it is an upper bound for any solution of OPT-EXT. So we have

$$\mathsf{Ext}_\pi(G) \leq T(G). \tag{7.6}$$

Now, consider the vertices to which the objects in allocation π are allocated. We will convert the instance of allocation π to S_U in the way described as follows. Let us consider k stars again, centered on the k vertices to which the objects with the k highest valuations are allocated under π, namely objects $o_1, o_2, ..., o_k$ in the aforementioned ordering. Let the new collection of stars be R_U. This means that $m - k$ objects are left over. Move these leftover objects to the leaves of these k stars (which include the neighbors of the centers in G) in an arbitrary way.

From the definition of S_U, we know that the total graph externality in S_U was:

$$U(G) = \sum_{i=1}^{k} (d_i \cdot o_i) - \sum_{j=k+1}^{m} o_j \tag{7.7}$$

Note that while S_U may not be a feasible externality from G, it is a graph instance using the object set O, therefore the graph externality of S_U is clearly lower than the trivial upper bound $T(G)$. Since the graph externality of S_U is $U(G)$, it follows that

$$U(G) \leq T(G) \tag{7.8}$$

Note that, in this scenario, if a vertex appears in both stars, we do not count it twice, but we only count it once (we may assume that it derives the highest externality out of its neighbors). This would result in some leaves of some stars without deriving externality. Note that if a vertex is both a center of a star and also in leaves of other stars, we will consider the vertex to appear in the center of the star in which it appears and not in the leaves. Since this would also mean that a vertex of V would appear in two or more stars in this scenario, we could have an object o_p not appearing at all in these k stars. Formally, the case for o_p is one of the following:

1. o_p is the center of another star, let it be star $k + 1$. This means that in S_U, o_p was deriving externality and now it is not, so the externality in this case is lower than o_p deriving externality from a star in S_U.

2. o_p is a leaf of another star, let it be star $k+1$. This means that another object, let it be o_{p+1}, is the center of star $k + 1$. This means that in S_U, o_{p+1} was deriving externality and now it is not, so the externality in this case is lower than o_{p+1} deriving externality from a star in S_U.

3. There exists no element o_p that does not belong in any of the k stars.

In the first two cases, we can see that graph externality is strictly lower than in the third case, since we could sequentially reallocate objects that do not belong in any of the k stars to "empty" vertices in the stars. This procedure would either increase the graph externality, when we reallocate a vertex that is a center of a star that is not in the k stars (since it will now derive externality instead of not deriving), and also when we reallocate a vertex that is a leaf of a star that is not in the k stars (since the objects in the centers of the k stars are the objects with maximum valuation, the externality the vertex will now derive will be higher than before). Therefore, by repeating this finite process until we cannot repeat it anymore, we get the third case with a not lower graph externality.

Having already all the objects within our k stars, with the k objects of highest valuation being in the centers of the stars, we observe that the graph externality of R_U is at most

$$\mathsf{Ext}_\pi(G) \leq \sum_{i=1}^{k} \left(o_i \cdot deg(\pi^{-1}(\nu(o_i)))\right) - \sum_{j=k+1}^{m} o_j, \tag{7.9}$$

with the equality holding iff we did not perform any reallocations beforehand.

We observe that we get the term $o_i \cdot deg(\pi^{-1}(\nu(o_i)))$ since we do not have a guarantee that the objects are placed to the stars in an order of non-increasing degree, therefore we do not know if o_1 is placed on v_1 (the vertex with degree d_1) and so on. We reallocate the objects $o_1, o_2, ..., o_k$ throughout the centers of the stars so that they are allocated to the same vertices as in S_U. We can observe that during these reallocations, the total graph externality can only increase, since we may move objects that are on centers to other centers of at least the same degree. Therefore, we have that the graph externality after the reallocations is at least as high as the graph externality before the reallocations. Formally, we get that

$$\sum_{i=1}^{k} \left(o_i \cdot deg(\pi^{-1}(\nu(o_i)))\right) - \sum_{j=k+1}^{m} o_j \leq \sum_{i=1}^{k} (d_i \cdot o_i) - \sum_{j=k+1}^{m} o_j. \tag{7.10}$$

But we observe that the right part of inequality 7.10 is equal to $U(G)$ (equality 7.7). This means that we have

$$\mathsf{Ext}_\pi(G) \leq U(G). \tag{7.11}$$

Combining inequalities 7.11 and 7.8, we get that

$$\mathsf{Ext}_\pi(G) \leq U(G) \leq T(G), \tag{7.12}$$

which concludes the proof.

■

We will move on to propose a greedy algorithm that computes a feasible solution for OPT-EXT in various classes of graphs. Since $U(G)$ is easy to compute, we can see that $U(G)$ is an upper bound to the optimal solution that can be used to measure how good the greedy algorithm performs, in terms of something better than the optimal solution.

The steps of the greedy algorithm are described in the description of Algorithm 7. Intuitively, Algorithm 7 considers the colorings blue-red-white in the same fashion they are presented in the analysis for OPT-EXT(0,1). Every time, the algorithm selects a non-blue vertex with the most white neighbors and allocates the highest valuation object to the blue vertex, whilst allocating the lowest valuation objects to the white vertices. In this manner, all the "low-valued" white vertices (which are now red) derive externality from the "high-valued" blue vertex.

Algorithm 7

Input: G, O

Output: An allocation π

```
1: Color every vertex of G white
2: Mark all objects in O as available
3: while there exist available objects in O do
4:    Let o be the object of largest valuation among available objects in O, and v be the
      vertex of G that has the largest number of white vertices in its closed neighborhood
      and is either red with a valuation ν(π(v)) < ν(o), or white
5:    if v is red then
6:       Mark object π(v) as available
7:    end if
8:    π(v) ← o;  Color v blue
9:    Mark object o as unavailable (allocated)
10:   while there exists a white neighbor w of v do
11:      Let o' be the object of smallest valuation among available objects in O
12:      π(w) ← o';  Color w red and mark o' as unavailable
13:   end while
14: end while
15: return π
```

The source code of Algorithm 7 is listed on Appendix A.1.

We ran Algorithm 7 on 10 DIMACS datasets from the 10th DIMACS Implementation challenge [14]. We selected valuations of objects that follow the uniform distribution, ranging from 0 to $4 \cdot |V|$.

Each benchmark used is presented in Table 7.1, along with its number of vertices $|V|$ and its number of edges $|E|$. We can see that these datasets are both sparse and dense.

In Figure 7.1, we can see the percentages of $U(G)$ achieved by running Algorithm 7 in the

Benchmark Characteristics			
ID	**Name**	**# Vertices**	**# Edges**
b1	karate	34	78
b2	dolphins	62	159
b3	lesmis	77	254
b4	adjnoun	112	425
b5	polbooks2	105	441
b6	chesapeake	39	170
b7	celegans_metabolic	453	2025
b8	football	115	613
b9	celegansneural	297	2148
b10	jazz	198	2742

Table 7.1: List of used benchmarks

benchmarks. We can observe that although $U(G)$ is clearly an overestimation of the optimal solution, Algorithm 7 provides solutions that differ no more than 5% from $U(G)$. Thus, the results of Algorithm 7 for the DIMACS benchmarks differ no more than 5% from the optimal solution.

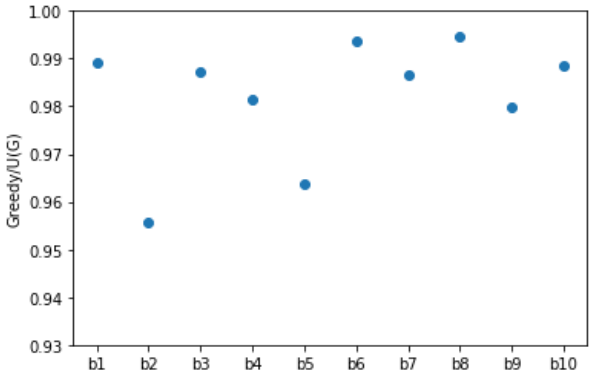


Figure 7.1: Percentage of $U(G)$ achieved by Algorithm 4

From the results of figure 7.1, we observe that there might be a correlation between the density of the graph and the achieved ratio of the externality obtained to $U(G)$. This can be explained from the fact that the denser a graph is, we will need a lower number of stars to cover S_U.

For instance, in the complete graph, we know that the optimal solution is achieved by having only 1 blue vertex, which could also be any vertex. In this case, we will also have that the optimal solution will be not only equal to $U(G)$, but also to $T(G)$. We therefore see that the lower the number of stars required to cover the whole set of vertices in S_U, the lower the "distortion" of the graph is in the result, and the higher is the likelihood that Algorithm 7.1 will provide a result with very few blue vertices (i.e. closer to a simple star partition).

Certain cases, such as the good result in the *karate* benchmark ($b1$), can also be explained. The Karate dataset is a fundamental example in community detection papers [22, 37, 46],

since it describes the case in which we have 2 very popular vertices, let them be v_1, v_2 that have almost every other vertex connected to them, but with the property that these vertices act as "leaders of different groups". This means that it is very likely that if a vertex v neighbors v_1, then probably v will not be a neighbor of v_2 as well. Simultaneously, it is likely that if two other vertices v_3, v_4 are connected with an edge, then both of them are probably connected to the same vertex from the set $\{v_1, v_2\}$. We can see that, despite the karate benchmark not being a dense graph, Algorithm 7 works well for this case, since it is the optimal scenario to allocate the highest valuation objects to vertices v_1 and v_2, having all the other vertices derive externality from the group leaders.

Focused even more on graph density, we computed several random graphs of different densities, so as to experimentally determine the performance of Algorithm 7. Let us define $h(G)$ as a density number, being the ratio of the edges of a graph to the edges of the full graph of the same number of vertices. Formally, $h(G)$ is defined as

$$h(G) := \frac{|E|}{\left(\frac{|V| \cdot (|V| - 1)}{2}\right)} \tag{7.13}$$

In Table 7.2, we observe the clear correlation between $h(G)$ and the ratio of the result of the Algorithm 7 to $U(G)$. Note that the graph created is not always the same — that is, the instance of 170 edges is not the instance of 150 edges with 20 more edges created, but a completely different random graph.

The values of Table 7.2 are some of the values generated from the code, which generates graphs of 150 vertices, for edges from 150 up to 11000 inclusive, with intervals of 20 edges. The complete results of the ratio are depicted as a graph in Figure 7.2, including the results of Table 7.2 but not only limited to them.

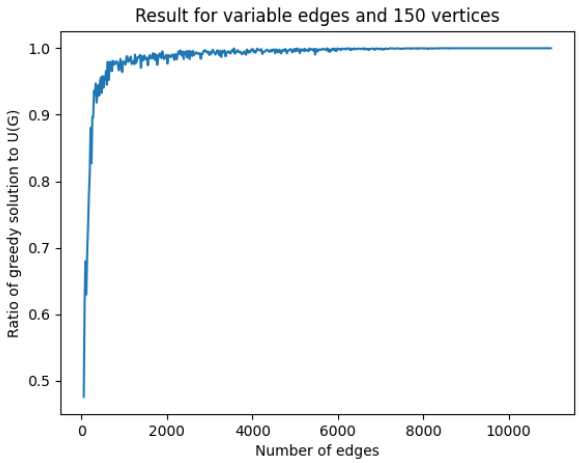


Figure 7.2: Density of random graphs and Greedy/$U(G)$ for 150 vertices

Random graph benchmarks of $\|V\| = 150$		
# Edges	$h(G)$	**Ratio of Algorithm 7 solution to $U(G)$**
150	0.0134	0.7303
170	0.0152	0.7788
190	0.0170	0.8127
210	0.0188	0.8805
230	0.0206	0.8270
250	0.0223	0.8967
270	0.0242	0.8963
290	0.0260	0.9358
390	0.0349	0.9446
690	0.0617	0.9795
1050	0.0940	0.9877
4290	0.3839	0.995
8050	0.7204	0.9999

Table 7.2: Results depending on density of graph

Chapter 8

Conclusions - Future Directions

Positive externality can be exerted from a neighbor to another, as a benefit from the highest valued neighbor to the lowest valued one. Many real-life examples, such as placing less popular posts in a news web page, can be modeled through an externality model in graphs. In our model, an object allocated to a vertex derives positive externality from at most one neighbor that must have a valuation higher than that of the object. Summing the externality of all vertices, we get the *graph externality*. The problem of maximizing the graph externality for specific graph, objects and valuations is **NP**-hard and is called OPT-EXT, which can be shown through proving a relation with the DOMINATING SET problem.

When the available valuations are only 0 or 1, the problem remains **NP**-hard and is called OPT-EXT(0,1). For this special case, there exists a constant approximation algorithm with a ratio of at least $(e-1)/(1+e) \approx 0.46$ and at most $2/3$. Since the externality function is shown not to be monotone submodular, greedy improvements of the $(e-1)/(1+e) \approx 0.46$ ratio are hindered. Through a connection with the problem of PARTIAL DOMINATING SET, we get exact algorithms for OPT-EXT(0,1) for graphs with treewidth of at most 10 and apex-minor-free graphs.

OPT-EXT is in **P** for graphs of degree at most 2, that is, when the graph is a collection of paths and cycles. A 0.5-approximation algorithm for caterpillar trees, a special case of trees, is also presented for OPT-EXT.

We present experimental results for the performance of a greedy algorithm for OPT-EXT. Running the algorithm on certain DIMACS benchmarks, we get that the greedy result is better than 95% of a nontrivial upper bound of the optimal solution. The algorithm has a near-perfect performance on denser graphs, since in this case, the upper bound is closer to the optimal solution.

As future work, attempts can be made to tighten the approximation ratio for the constant approximation algorithm for OPT-EXT(0,1), probably using a different approach. A lower bound for the greedy algorithm used for the experimental evaluation could also be calculated. Additionally, more variants of the problem can be examined, such as deriving externality from more neighbors, or deriving externality from non-neighbors, using attenuation factors. Lastly, variations of the model could be tested in auctions, where buyers make offers for objects, or where the buyers are the objects and make offer for a desirable placement in the graph.

Appendix A

Appendix — Source code

A.1 C++ code for externality of DIMACS benchmarks

The C++ code for the calculation of the performance of DIMACS benchmarks follows. The benchmark "chesapeake.graph" is the one included in the code by default, but the same code applies for the other benchmarks.

```
#include <bits/stdc++.h>
#include <sys/time.h>
using namespace std;

#define int long long
#define ll long long
#define rep(i,n) for (int i = 0; i < (n); i++)
#define rrep(i,n) for (int i = (n)-1; i >= 0; i--)
#define rap(i,a,n) for (int i = a; i < (n); i++)
#define rrap(i,n,a) for (int i = (n)-1; i >= a; i--)
#define LL_MAX 9223372036854775807

typedef vector<int> vi;
typedef pair<int,int> ii;
typedef vector<ii> vii;
typedef list<int> li;
typedef unordered_map<int,int> mii;

vector<vi> adj, adj_app;
vi color1, color2, visible, degrees, degrees_unsorted;

int32_t main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    ifstream fin;
    ofstream fout;
    fin.open("chesapeake.graph");
    fout.open("sbm1.out");

    int n,m,x;
    fin >> n >> m >> x;
    string pp;
```

```
    getline(fin,pp);

    adj.assign(n,vi());
    adj_app.assign(n,vi());
    color1.assign(n,0); //0: white, 1: blue, 2: red, initially all vertices white
    color2.assign(n,0); //color2 is for the variant with hidden vertices
    visible.assign(n,0); //0: hidden, 1: visible

    //create the initial array (adj)
    rep(i,n) {
        getline(fin,pp);
        stringstream ss(pp);
        int nn;
        while (ss >> nn) {
            adj[i].push_back(nn-1);
        }
    }

    //generate randomly n valuations for objects, between 0 and 4*n, inclusive,
    //sorting the resulting array decreasingly
    int gen;
    srand(time(NULL)); //random number depending on second of day
    vi vals;
    rep(i,n) {
        gen = rand()%(4*n + 1);
        vals.push_back(gen);
    }
    sort(vals.begin(),vals.end(),greater<int>());

    //calculate the degrees for each vertex and sort them in
    //non-increasing order, so that we calculate the upper bounds
    rep(i,n) {
        degrees_unsorted.push_back(adj[i].size());
    }
    degrees = degrees_unsorted;
    sort(degrees.begin(),degrees.end(),greater<int>());

    int i1 = 0, i2 = n-1, su = 0;
    while (i1 < i2) {
        int cur = vals[i1];
        int deg = degrees[i1];
        rep(j,deg) {
            su += (cur - vals[i2]);
            i2--;
            if (i1 >= i2) break;
        }
        if (i1 >= i2) break;
        i1++;
    }

    int res0 = su; //res0 is the upper bound for the algorithm

    //solve the problem normally
    vi avail1; //available valuation
    avail1.assign(n,1);
```

```
    vi val1; //pointer of valuation allocated to vertex
    val1.assign(n,-1);
    rep(i,n) {
        if (!avail1[i]) continue;
        //find the vertex that is red with a smaller valuation than the
        //next largest one or white with the most white neighbors
        int mx = -1; //the number of white neighbors
        int mxv = -1; //the vertex
        rep(j,n) {
            int cnt = 0;
            if (color1[j] == 1 || (color1[j] == 2 &&
            vals[val1[j]] >= vals[i])) continue;
            for (auto x: adj[j]) {
                if (color1[x] == 0) cnt++;
            }
            if (cnt > mx) {
                mx = cnt;
                mxv = j;
            }
        }
        if (color1[mxv] == 2) avail1[val1[mxv]] = 1;
        color1[mxv] = 1;
        val1[mxv] = i;
        avail1[i] = 0;
        if (mxv == -1) break;
        for (auto x: adj[mxv]) {
            if (color1[x] == 0) {
                color1[x] = 2;
                rrep(k,n) {
                    if (avail1[k]) {
                        avail1[k] = 0;
                        val1[x] = k;
                        break;
                    }
                }
            }
        }
    }
    //calculate total externality
    int res1 = 0;
    rep(i,n) {
        int mx = vals[val1[i]];
        for (auto x: adj[i]) {
            if (vals[val1[x]] > mx) mx = vals[val1[x]];
        }
        if (mx > vals[val1[i]]) {
            res1 += mx - vals[val1[i]];
        }
    }

    fout << res0 << ' ' << res1 << '\n';

    return 0;
}
```

A.2 C++ code for the generation of dense graphs

The C++ code for the generation of the dense graphs follows. Albeit the calculation of the graph externality is the same as before, the whole code is being presented, so that the program can be instantly compiled.

```
#include <bits/stdc++.h>
using namespace std;

#define int long long
#define ll long long
#define rep(i,n) for (int i = 0; i < (n); i++)
#define rrep(i,n) for (int i = (n)-1; i >= 0; i--)
#define rap(i,a,n) for (int i = a; i < (n); i++)
#define rrap(i,n,a) for (int i = (n)-1; i >= a; i--)
#define LL_MAX 9223372036854775807

typedef vector<int> vi;
typedef pair<int,int> ii;
typedef vector<ii> vii;
typedef list<int> li;
typedef unordered_map<int,int> mii;

int myrandom (int i) { return std::rand()%i;}

int32_t main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    srand(time(NULL));

    ofstream fout;
    fout.open("dense0.txt");

    int n,x;
    x = 0;
    n = 150;
    int cnt00 = 0;

    for (int m = 50; m <= 11000; m += 20) {
        cnt00++;
        vi shuf;
        vii edg;
        vector<vi> adj;
        adj.assign(n,vi());

        rep(i,n*(n-1)/2) {
            shuf.push_back(i);
        }

        rep(i,n) {
            rap(j,i+1,n) {
                edg.push_back(ii{i,j});
            }
```

```
    }

    random_shuffle(shuf.begin(),shuf.end(),myrandom);

    //creating random graph
    rep(i,m) {
        int ind = shuf[i];
        ii x = edg[ind];
        int p1 = x.first;
        int p2 = x.second;
        adj[p1].push_back(p2);
        adj[p2].push_back(p1);
    }

    vector<vi> adj_app;
    vi color1;
    adj_app.assign(n,vi());
    color1.assign(n,0); //0: white, 1: blue, 2: red,
    //initially all vertices white

     //generate randomly n valuations for objects, between 0 and 4*n,
     //inclusive, sorting the resulting array decreasingly
    int gen;
    vi vals;
    rep(i,n) {
        gen = rand()%(4*n + 1);
        vals.push_back(gen);
    }
    sort(vals.begin(),vals.end(),greater<int>());

    //calculate the degrees for each vertex and sort them in
    //non-increasing order, so that we calculate the upper bounds
    vi degrees_unsorted;
    vi degrees;
    rep(i,n) {
        degrees_unsorted.push_back(adj[i].size());
    }
    degrees = degrees_unsorted;
    sort(degrees.begin(),degrees.end(),greater<int>());

    int i1 = 0, i2 = n-1, su = 0;
    while (i1 < i2) {
        int cur = vals[i1];
        int deg = degrees[i1];
        rep(j,deg) {
            su += (cur - vals[i2]);
            i2--;
            if (i1 >= i2) break;
        }
        if (i1 >= i2) break;
        i1++;
    }

    int res0 = su; //res0 is the upper bound for the algorithm
```

```
//solve the problem normally
vi avail1; //available valuation
avail1.assign(n,1);
vi val1; //pointer of valuation allocated to vertex
val1.assign(n,-1);
rep(i,n) {
    if (!avail1[i]) continue;
    //find the vertex that is red with a smaller valuation than the
    //next largest one or white with the most white neighbors
    int mx = -1; //the number of white neighbors
    int mxv = -1; //the vertex
    rep(j,n) {
        int cnt = 0;
        if (color1[j] == 1 || (color1[j] == 2 &&
        vals[val1[j]] >= vals[i])) {
            continue;
        }
        for (auto x: adj[j]) {
            if (color1[x] == 0) cnt++;
        }
        if (cnt > mx) {
            mx = cnt;
            mxv = j;
        }
    }
    if (color1[mxv] == 2) avail1[val1[mxv]] = 1;
    color1[mxv] = 1;
    val1[mxv] = i;
    avail1[i] = 0;
    if (mxv == -1) break;
    for (auto x: adj[mxv]) {
        if (color1[x] == 0) {
            color1[x] = 2;
            rrep(k,n) {
                if (avail1[k]) {
                    avail1[k] = 0;
                    val1[x] = k;
                    break;
                }
            }
        }
    }
}
//calculate total externality
int res1 = 0;
rep(i,n) {
    int mx = vals[val1[i]];
    for (auto x: adj[i]) {
        if (vals[val1[x]] > mx) mx = vals[val1[x]];
    }
    if (mx > vals[val1[i]]) {
        res1 += mx - vals[val1[i]];
    }
}
```

```
        fout << n << ' ' << m << ' ' << res0 << ' ' << res1 << ' ' <<
        ((float)res1)/res0 << '\n';
    }

    return 0;
}
```

Bibliography

[1] R. Abebe, J. M. Kleinberg, and D. C. Parkes. Fair division via social comparison. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 281–289, 2017.

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

[3] N. AhmadiPourAnari, S. Ehsani, M. Ghodsi, N. Haghpanah, N. Immorlica, H. Mahini, and V. S. Mirrokni. Equilibrium pricing with positive externalities. *Theor. Comput. Sci.*, 476:1–15, 2013.

[4] S. Athey and G. Ellison. Position auctions with consumer search. *The Quarterly Journal of Economics*, 126(3):1213–1270, 2011.

[5] A. Beynier, Y. Chevaleyre, L. Gourvès, J. Lesca, N. Maudet, and A. Wilczynski. Local envy-freeness in house allocation problems. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 292–300, 2018.

[6] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, editors. *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

[7] S. Brânzei, T. P. Michalak, T. Rahwan, K. Larson, and N. R. Jennings. Matchings with externalities and attitudes. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 295–302, 2013.

[8] S. Brânzei, A. D. Procaccia, and J. Zhang. Externalities in cake cutting. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 55–61, 2013.

[9] R. Bredereck, A. Kaczmarczyk, and R. Niedermeier. Envy-free allocations respecting social networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS-18)*, pages 283–291, Stockholm, Sweden, July 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[10] G. J. Chang. Algorithmic aspects of domination in graphs. Technical report, Department of Mathematics, National Taiwan University, 2011. Available at http://www.math.ntu.edu.tw/∼mathlib/preprint/2011-01.pdf.

[11] M. Chlebík and J. Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Inf. Comput.*, 206(11):1264–1275, 2008.

[12] E. Cockayne, S. Goodman, and S. Hedetniemi. A linear algorithm for the domination number of a tree. *Information Processing Letters*, 4(2):41 – 44, 1975.

[13] E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[14] Dimacs. www.cc.gatech.edu/dimacs10/archive/clustering.shtml.

[15] F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Subexponential algorithms for partial cover problems. *Inf. Process. Lett.*, 111(16):814–818, 2011.

[16] D. Fotakis, L. Gourvès, S. Kasouridis, and A. Pagourtzis. Object allocation and positive graph externalities. *ECAI 2020*, 2020.

[17] D. Fotakis, P. Krysta, and O. Telelis. Externalities among advertisers in sponsored search. In *Algorithmic Game Theory, 4th International Symposium, SAGT 2011, Amalfi, Italy, October 17-19, 2011. Proceedings*, pages 105–116, 2011.

[18] D. Fotakis and P. Siminelakis. On the efficiency of influence-and-exploit strategies for revenue maximization under positive externalities. *Theor. Comput. Sci.*, 539:68–86, 2014.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[20] M. Ghodsi, H. Saleh, and M. Seddighin. Fair allocation of indivisible items with externalities. *CoRR*, abs/1805.06191, 2018.

[21] A. Ghosh and M. Mahdian. Externalities in online advertising. In *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 161–168, 2008.

[22] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[23] L. Gourvès, J. Lesca, and A. Wilczynski. Object allocation via swaps along a social network. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 213–219, 2017.

[24] L. Gourvès, J. Monnot, and L. Tlilane. Approximate tradeoffs on weighted labeled matroids. *Discrete Applied Mathematics*, 184:154–166, 2015.

[25] N. Haghpanah, N. Immorlica, V. S. Mirrokni, and K. Munagala. Optimal auctions with positive network externalities. In *Proceedings 12th ACM Conference on Electronic Commerce (EC-2011)*, pages 11–20, 2011.

[26] K. Haraguchi. An efficient local search for the minimum independent dominating set problem. In *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, pages 13:1–13:13, 2018.

[27] F. Harary and A. J. Schwenk. The number of caterpillars. *Discrete Mathematics*, 6(4):359 – 365, 1973.

[28] J. Hartline, V. S. Mirrokni, and M. Sundararajan. Optimal marketing strategies over social networks. In *Proceedings 17th Conference on World Wide Web (WWW-2008)*, pages 189–198, 2008.

[29] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, editors. *Domination in graphs: Advanced Topics*. Marcel Dekker Inc., New York, NY, USA, 1998.

[30] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, MA, USA, 1997.

[31] S. Huang and M. Xiao. Object reachability via swaps along a line. In *Proceedings of the The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19), Honolulu, Hawaii, USA, January 27 – February 1, 2019*, pages x–y, 2019. To appear.

[32] P. Hummel and R. P. McAfee. Position auctions with externalities. In *Web and Internet Economics - 10th International Conference, WINE 2014, Beijing, China, December 14-17, 2014. Proceedings*, pages 417–422, 2014.

[33] M. L. Katz and C. Shapiro. Network externalities, competition, and compatibility. *The American Economic Review*, 75(3):424–440, 1985.

[34] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 137–146, 2003.

[35] L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53 – 72, 1980.

[36] P. Krysta, T. P. Michalak, T. Sandholm, and M. Wooldridge. Combinatorial auctions with externalities. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 1471–1472, 2010.

[37] K. Kulkarni, A. Pagourtzis, K. Potika, P. Potikas, and D. Souliou. Community detection via neighborhood overlap and spanning tree computations. In *Algorithmic Aspects of Cloud Computing - 4th International Symposium, ALGOCLOUD 2018*, volume 11409 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2018.

[38] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930.

[39] R. Paes Leme, V. Syrgkanis, and É. Tardos. Sequential auctions and externalities. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 869–886, 2012.

[40] M. Li, J. Zhang, and Q. Zhang. Truthful cake cutting mechanisms with externalities: Do not make them care for others too much! In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 589–595. AAAI Press, 2015.

[41] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978.

[42] C. T. Nguyen, J. Shen, M. Hou, L. Sheng, W. Miller, and L. Zhang. Approximating the spanning star forest problem and its application to genomic sequence alignment. *SIAM J. Comput.*, 38(3):946–962, 2008.

[43] E. Pataki, A. Sagi, and K. Jozef. Externalities and the optimal allocation of economic resources. In *2015 IEEE 13th International Symposium on Intelligent Systems and Informatics (SISY)*, pages 185–188, Sep. 2015.

[44] J. Rohlfs. A theory of interdependent demand for a communications service. *The Bell Journal of Economics and Management Science*, 5(1):16–37, 1974.

[45] M. Seddighin, H. Saleh, and M. Ghodsi. Externalities and fairness. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, pages 538–548, 2019.

[46] D. Souliou, P. Potikas, K. Potika, and A. Pagourtzis. Weight assignment on edges towards improved community detection. In *Proceedings of the 23rd International Database Applications & Engineering Symposium, IDEAS 2019*, pages 3:1–3:5. ACM, 2019.

[47] G. Steiner. On the k-path partition of graphs. *Theor. Comput. Sci.*, 290(3):2147–2155, 2003.

[48] R. A. Velez. Fairness and externalities. *Theoretical Economics*, 11:381–410, 2016.

[49] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1514–1522, 2018.