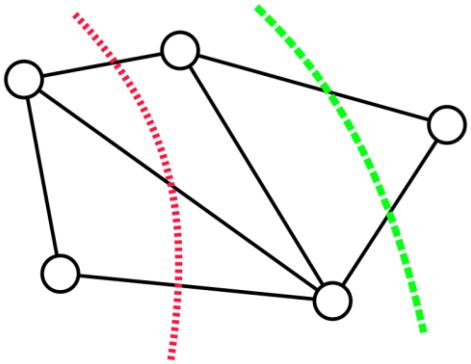
Υλοποίηση Αλγορίθμου Stoer-Wagner

Αντωνίου Δημήτρης

Διπλωματική Εργασία



Επιβλέπων: Λουκάς Γεωργιάδης

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Ευχαριστίες

Έχοντας ολοκληρώσει την διπλωματική μου εργασία θα ήθελα να ευχαριστήσω θερμά τον επιβλέπον καθηγητή μου κ. Λ. Γεωργιάδη που με την απέραντη στήριξη του με βοήθησε καθόλα την διάρκεια της πραγματοποίησης της διατριβής μου. Επίσης , θα ήθελα να ευχαριστήσω την οικογένεια μου, τους φίλούς μου αλλά και πιο προσωπικά την μητέρα μου Ζωή και τον αδερφό μου Σωτήρη για την υποστήριξη τους προς το πρόσωπο μου καθόλα την διάρκεια της ακαδημαϊκής μου σταδιοδρομίας. Η διπλωματική μου εργασία είναι αφιερωμένη στην μνήμη του πατέρα μου Κωνσταντίνου Αντωνίου που έφυγε πρόωρα από την ζωή και ήταν ο πρώτος άνθρωπος που με πίστεψε και με καθοδήγησε να υπηρετήσω την πληροφορική.

Ιωάννινα Φεβρουάριος 2023

Αντωνίου Δημήτρης

Περίληψη στα ελληνικά

Ένα από τα πιο χρήσιμα γνωστικά πεδία στην σύγχρονη επιστήμη των υπολογιστών είναι η θεωρία γραφημάτων. Τα γραφήματα είναι διακριτές δομές, οι οποίες αποτελούνται από κορυφές και ακμές οι οποίες συνδέουν αυτές τις κορυφές. Στην σήμερον ημέρα τα γραφήματα διέπουν ένα σημαντικό μέρος της καθημερινότητα μας χωρίς να μας γίνεται ιδιαίτερα αντιληπτό, και όμως τα γραφήματα έχουν πολλές λειτουργίες όπως ένα δίκτυο υπολογιστών, ένα κοινωνικό δίκτυο την μοντελοποίηση των χαρτών αλλά ακόμα και στις αναθέσεις εργασιών μίας εταιρείας. Στην παρούσα διπλωματική εργασία μελετούμε ενδελεχώς το πρόβλημα της ελάχιστης τομής μέσω της ανάλυσης και της υλοποίησής του αλγόριθμου Stoer & Wagner, σε συντομία *SW,* από την αγγλική βιβλιογραφία, όπου είναι και ένας από τους ελάχιστους αλγορίθμους που χρησιμοποιούν την τεχνική συρρίκνωσης των ακμών ενός γραφήματος, για τον υπολογισμό τις ελάχιστης τομής. Παράλληλα, με αφετηρία τον αλγόριθμο *SW* αναλύεται και ο αλγόριθμος Ford Fulkerson οπού ο συγκεκριμένος αλγόριθμος ανήκει σε μια κατηγορία αλγορίθμων που υπολογίζει την ελάχιστη τομή ενός γραφήματος με τεχνικές ροής. Προς το τέλος της διατριβής γίνονται και κάποια πειράματα που αναλύουν και συγκρίνουν την πολυπλοκότητα των δύο αλγορίθμων.

Λέξεις Κλειδιά: Γράφημα , Ελάχιστη Τομή , Ροή Δικτύου, Stoer Wagner , Ford-Fulkerson, Python

Abstract

One of the most useful fields in modern computer science is graph theory. Graphs are discrete structures, consisting of vertices and edges that connect these vertices. In the present day, graphs govern an important part of our daily life without being a particularly.

aware of it, and nevertheless graphs have many functions such as a computer network, a social network, the modelling of maps and even in the assignment of tasks of a company. In this diplomatic thesis we study in detail the problem of minimum cut through the analysis and implementation of Stoer & Wagner algorithm, in short SW, from the English literature where it is one of the minimal algorithms that use the technique of shrinking the edges of a graph to compute the minimum cut. At the same time, starting from SW, the Ford Fulkerson algorithm is also analyzed where this algorithm belongs to a class of algorithms that computes the minimum cut of a graph using flow techniques. Toward the end of the diplomatic thesis some experiments are also done to analyze and compare the complexity of the two algorithms.

Keywords: Graph, Minimum Cut, Network Flow, Stoer Wagner, Ford Fulkerson, Python

Πίνακας περιεχομένων

Κατάλογος Σχημάτων

Κεφάλαιο 1. Εισαγωγή

Καθώς η τεχνολογία αναπτύσσεται με γοργούς ρυθμούς καθημερινά προκύπτουν νέα προβλήματα, πολλά από τα οποία έχουν να κάνουν με την διαχείριση και την ανάλυση γραφημάτων. Επομένως στην περιοχή της θεωρίας αλγορίθμων σε γραφήματα αναζητούμε όλο και καλύτερους τρόπους ανάλυσης και επεξεργασίας γραφημάτων έτσι ώστε να μπορούν να χειριστούν γραφήματα ευρείας κλίμακας. Η εφαρμογή κλασικών αλγορίθμων στο πεδίο γραφημάτων γίνεται ολοένα και πιο δύσκολη καθώς όσο αυξάνονται οι διαστάσεις ενός γραφήματος όπως κόμβοι ή ακμές αυξάνεται και η πολυπλοκότητα αλλά και το υπολογιστικό κόστος. Για αυτό το λόγο ο στόχος των αλγορίθμων στο πεδίο γραφημάτων στην εποχή μας είναι να βρει λύσεις στα σύγχρονα προβλήματα της πληροφορικής.

1.1 Αντικείμενο της διπλωματικής

Το αντικείμενο της διπλωματικής εργασίας είναι η υλοποίηση του αλγορίθμου ελάχιστης τομής Stoer-Wagner στην γλώσσα προγραμματισμού python σε σύγκριση με αλγόριθμους που χρησιμοποιούν τεχνικές ροής και πιο συγκεκριμένα τον αλγόριθμο Ford-Fulkerson, με τέτοιο τρόπο ώστε να καθίσταται δυνατή η άντληση συμπερασμάτων για τους αλγορίθμους *(SW)* και Ford-Fulkerson. Οι απεικονίσεις και τα πειράματα που χρησιμοποιούνται είναι σχεδιασμένες με τέτοιο τρόπο ώστε να προσφέρουν άμεσα την ικανότητα στον αναγνώστη να μπορεί να αντλεί πληροφορίες οι οποίες είναι ΄΄κρυμμένες΄΄ πίσω από τα γραφήματα και τις γραφικές παραστάσεις. Παράλληλα η εργασία πραγματεύεται την ανάλυση της πολυπλοκότητας του αλγόριθμου τις μεθόδους που χρησιμοποιήσαμε για την υλοποίηση, αλλά και την βελτιστοποίηση του αλγόριθμου (SW) τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο.

1.2 Οργάνωση της διπλωματικής εργασίας

Η συγκεκριμένη διπλωματική εργασία αποτελείται από 6 κεφάλαια τα οποία αναλύονται στις παρακάτω παραγράφους ενδελεχώς.

1. Στο Κεφάλαιο 1 γίνεται μια ιστορική αναδρομή και περιγράφονται κάποιες βασικές έννοιες των γραφημάτων. Επιπρόσθετα γίνεται μια εμπεριστατωμένη εισαγωγή στις ροές και στις τομές ενός γραφήματος.
2. Στο Κεφάλαιο 2 αναφερόμαστε στο πρώτο σκέλος του υπολογισμού της ελάχιστής τομής, με τεχνικές ροής ενώ παράλληλα γίνεται μια εκτενής ανάλυση του αλγορίθμου Ford Fulkerson.
3. Στο Κεφάλαιο 3 αναφερόμαστε στο βασικό αλγόριθμο εύρεσης ελάχιστής τομής Stoer-Wagner οπού γίνεται η ανάλυση του τόσο σε θεωρητικό επίπεδο όσο και σε πρακτικό επίπεδο.
4. Στο Κεφάλαιο 4 παραθέτουμε εκτενώς την περιγραφή των υλοποιήσεων για κάθε αλγόριθμο που αναλύσαμε στις προηγούμενες ενότητες.
5. Στο Κεφάλαιο 5 παραθέτουμε τα πειράματα για κάθε αλγόριθμο και τα αξιολογούμε με βάση τα αποτελέσματα τους.
6. Στο Κεφάλαιο 6 συνοψίζουμε τα συμπεράσματα της διατριβής μας.

1.3 Ιστορική αναδρομή

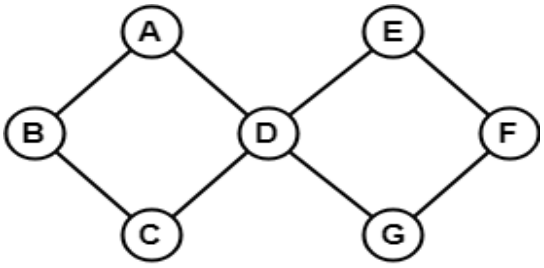
Η θεωρία γραφημάτων είναι ένας κλάδος των μαθηματικών που μελετάει τις ιδιότητες των γραφημάτων και έχει άμεση σχέση με την περιοχή της θεωρίας αλγορίθμων που πραγματεύονται την ανάλυση γραφημάτων. Σημείο σταθμός για την θεμελίωση της θεωρίας γραφημάτων ως μια ξεχωριστής επιστήμης θεωρείται η σημαντική μελέτη του Λέοναρντ Όιλερ για τις επτά γέφυρες του Κένιγκσμπεργκ *(σημερινό καλίνιγκραντ)* το 1736. Μετέπειτα διάφοροι επιστήμονες από πολλά γνωστικά πεδία ασχοληθήκανε ενδελεχώς με την θεωρία γραφημάτων επί γραμματικά θα αναφέρουμε τον Γάλλο χημικό Αλεξάντρ-Τεοφίλ Βαντερμόντ όπου ασχολήθηκε με το κλασσικό μαθηματικό πρόβλημα του Ίππου στην σκακιέρα που κατευθύνεται με την ανάλυση θέσης, ο Γάλλος μαθηματικός Ωγκυστέν-Λουί Κωσύ που ασχολήθηκε σχετικά με τον αριθμό των ακμών, των κορυφών και των εδρών ενός κυρτού πολύεδρου. Τόσο κατά την διάρκεια του 19 αιώνα όσο και την διάρκεια του 20 αιώνα μέχρι και σήμερα πολλοί επιστήμονες θεμελίωσαν την θεωρία γραφημάτων ως ένα σημαντικό πεδίο τόσο της πληροφορικής όσο και των μαθηματικών.

1.4 Θεωρητικό υπόβαθρο

Στην υποενότητα αυτήν παρουσιάζονται αναλυτικά βασικές έννοιες των γραφημάτων που έχουν σχέση με την εργασία και είναι χρήσιμες για την κατανόηση του αναγνώστη στην ανάλυση των αλγορίθμων.

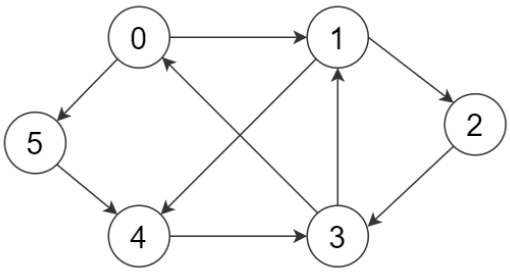
1.4.1 Γραφήματα

Ορισμός 1.1 Ένα γράφημα G=(V,E) αποτελείται από το V, ένα μη κενό σύνολο κορυφών (ή κόμβων) και από το E, ένα σύνολο ακμών. Κάθε ακμή έχει είτε μία είτε δύο κορυφές που σχετίζονται με αυτήν και ονομάζονται άκρα της ακμής. Μια ακμή θεωρείται ότι συνδέει τα άκρα της. Επίσης οι ακμές ενός γραφήματος μπορεί να έχουν κατευθύνσεις είτε να μην έχουν δηλαδή να είναι μη κατευθυνόμενα. Στην συγκεκριμένη διατριβή θα ασχοληθούμε με μη κατευθυνόμενα γραφήματα κατά κύριο λόγο. Επιπρόσθετα ένα γράφημα μπορεί να επιτρέψει να έχει πολλές παράλληλες ακμές δηλαδή να έχει την ιδιότητα ενός πολυγραφήματος και η κάθε ακμή να διαθέτει βάρη.



Σχήμα 1.1 Μη κατευθυνόμενο γράφημα

Ορισμός 1.2 Ένα κατευθυνόμενο γράφημα (V,E) αποτελείται από ένα μη κενό σύνολο κορυφών V και από ένα σύνολο κατευθυνόμενων ακμών E. Κάθε κατευθυνόμενη ακμή σχετίζεται με ένα διατεταγμένο ζεύγος κορυφών. Η κατευθυνόμενη ακμή που σχετίζεται με το διατεταγμένο ζεύγος (u, v) λέμε ότι ξεκινά από την κορυφή u και τερματίζει στην κορυφή v.

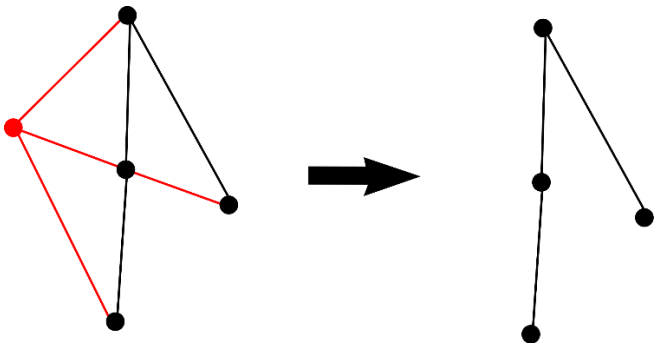


Σχήμα 1.2 Ένα κατευθυνόμενο γράφημα

Ορισμός 1.3 Ένας βρόχος σε ένα τυχαίο γράφημα G είναι μια ακμή (κατευθυνόμενη ή μη), η οποία αρχίζει και τελειώνει στην ίδια κορυφή. Στο πλαίσιο της διπλωματικής μας διατριβής οι βρόχοι δεν έχουν ιδιαίτερη σημασία.

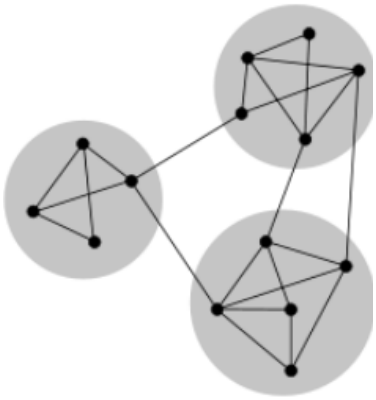
Ορισμός 1.4 Ο βαθμός μιας κορυφής σε ένα μη κατευθυνόμενο γράφημα είναι το πλήθος των ακμών που προσπίπτουν σε αυτή. Ο βαθμός της κορυφής δηλώνεται ως deg(v).

Ορισμός 1.5 Ένα υπογράφημα ενός γραφήματος G=(V,E) είναι ένα γράφημα H=(W,F), όπου το W υποσύνολο του V και F υποσύνολο του E. Ένα υπογράφημα H του G είναι ένα γνήσιο υπογράφημα του G, αν H≠G.



Σχήμα 1.3 Ένα τυχαίο γράφημα G με το υπογράφημα του H

Ορισμός 1.6 Ένα μη κατευθυνόμενο γράφημα ονομάζεται συνεκτικό, αν υπάρχει ένα μονοπάτι ανάμεσα σε κάθε ζεύγος διαφορετικών κορυφών του γραφήματος. Ένα μη κατευθυνόμενο γράφημα το οποίο δεν είναι συνεκτικό ονομάζεται μη συνεκτικό. Μπορούμε να πούμε ότι άμα αποσυνδέσουμε ένα γράφημα , όταν αφαιρούμε τις κορυφές ή τις ακμές ή και τα δύο, για να παράγουμε ένα μη συνεκτικό γράφημα.



Σχήμα 1.4 Συνεκτικό γράφημα

1.4.2 Προγραμματιστικά Εργαλεία

Η Python είναι μια διερμηνευόμενη *(interpreted),* γενικού σκοπού και υψηλού επιπέδου, γλώσσα προγραμματισμού. Ανήκει στις γλώσσες του προστακτικού προγραμματισμού *(imperative programming)* και υποστηρίζει τόσο το διαδικαστικό *(procedural programming)* όσο και το αντικειμενοστραφές *(object-oriented programming)* προγραμματιστικό υπόδειγμα. Παράλληλα είναι δυναμική γλώσσα προγραμματισμού *(dynamically typed)* και υποστηρίζει την συλλογή απορριμμάτων (*garbage collection ή GC*). Ο κύριος στόχος της είναι η αναγνωσιμότητα του κώδικά της και η ευκολία χρήσης της. Το συντακτικό της επιτρέπει στους προγραμματιστές να εκφράσουν έννοιες σε λιγότερες γραμμές κώδικα από ότι θα ήταν δυνατόν σε γλώσσες όπως η C++ ή η Java. Διακρίνεται λόγω του ότι έχει πολλές βιβλιοθήκες που διευκολύνουν ιδιαίτερα αρκετές συνηθισμένες εργασίες και για την ταχύτητα εκμάθηση της. Μειονεκτεί στο ότι επειδή είναι διερμηνευόμενη είναι πιο αργή από τις μεταγλωττιζόμενες *(compiled)* γλώσσες όπως η C και η C++. Για αυτό το δεν είναι κατάλληλη για γραφή λειτουργικών συστημάτων. Είναι σημαντικό να αναφέρουμε ότι η Python είναι γλώσσα ανοικτού κώδικά *(open source)* , ο κώδικα της δηλαδή είναι διαθέσιμος σε όλους και χωρίς κάποιο κόστος. Το λογισμικό ανοικτού κώδικα μπορεί ο καθένας ελεύθερα να το χρησιμοποιεί, να το αντιγράφει, να το διανέμει και να το τροποποιεί ανάλογα με τις ανάγκες που έχει θέσει .

Το PyCharm είναι ένα περιβάλλον ανάπτυξης από την JetBrains για την Python *(IDE)* όπου μπορούμε να γράψουμε σενάρια (*scripts)* καθώς τα εκτελούμε . Τα εργαλεία που μας προσφέρει είναι βοήθεια και ανάλυση κωδικοποίησης, με συμπλήρωση κώδικα, σύνταξη και επισήμανση σφαλμάτων. Παράλληλα μας δίνει την δυνατότητα να πλοηγούμαστε στο έργο και στο κώδικα , σε εξειδικευμένες προβολές έργου , προβολές δομής αρχείων και γρήγορη μετάβαση μεταξύ αρχείων, τάξεων , μεθόδων και χρήσεων. Το PyCharm δεν λειτουργεί μόνο για αρχεία python αλλά και υποστηρίζει και άλλες γλώσσες προγραμματισμού όπως JavaScript, kotlin ή CoffeeScript και άλλα εργαλεία όπως html ή CSS. Αυτό το καθιστά ένα πολύ αξιόπιστο προγραμματιστικό εργαλείο.

1.5 Εισαγωγή στα δίκτυα ροής

Στην υποενότητα αυτήν παρατίθεται ένας θεωρητικός ορισμός των δικτύων ροής, αναλύονται οι ιδιότητες τους, και ορίζεται σε μια τυπική μορφή το πρόβλημα της

μέγιστης ροής. Επιπρόσθετα γίνεται μια εισαγωγή σε ορισμένους χρήσιμους συμβολισμούς.

Ορισμός 1.7 Ένα δίκτυο ροής G=(V,E) είναι ένα κατευθυνόμενο γράφημα στο οποίο κάθε ακμή (u, υ) ∈ στο σύνολο E και έχει μια μη αρνητική χωρητικότητα c(u, υ)>=0. Μια συνθήκη που πρέπει επίσης να ισχύει είναι ότι αν το E περιέχει κάποια ακμή (u, υ). Στην περίπτωση όπου μια ακμή (u, υ) δεν ανήκει στο σύνολο ακμών ενός γραφήματος τότε ορίζουμε χάριν απλότητας ότι c(u, υ)=0, και απαγορεύουμε τους βρόχους. Εκείνο που έχει ιδιαίτερη σημασία είναι ότι σε ένα δίκτυο ροής υπάρχουν δύο ''ειδικοί'' κόμβοι : ένας αφετηριακός κόμβος s *(η ''πηγή'')* και ένας τερματικός κόμβος t *(η ''καταβόθρα'').* Χάριν απλότητας, υποθέτουμε επίσης ότι κάθε κόμβος βρίσκεται σε κάθε διαδρομή από τον αφετηριακό κόμβο προς τον τερματικό κόμβο . Δηλαδή για κάθε κόμβο u ανήκει στο σύνολο κορυφών V, το δίκτυο ροής περιέχει κάποια διαδρομή από το s προς το u και προς το t. Επομένως η διαπίστωση αύτη μας οδηγεί στο συμπέρασμα ότι το γράφημα είναι συνεκτικό , και δεδομένου του παραπάνω συμπεράσματος ο κάθε κόμβος εκτός του s έχει μία τουλάχιστον ακμή, |E|>=|V|-1. Κατόπιν αυτών, μπορούμε πλέον να δώσουμε έναν πιο τυπικό ορισμό της έννοιας της ροής . Έστω λοιπόν ότι έχουμε ένα δίκτυο ροής G=(V,E) με συνάρτηση χωρητικότητας c. Έστω s ο αφετηριακός κόμβος του δικτύου και t ο τερματικός κόμβος. Ροή στο γράφημα G είναι μια συνάρτηση f : V * V -> R η οποία παίρνει πραγματικές τιμές και ικανοποιεί τις εξής δύο ιδιότητες:

Περιορισμός χωρητικότητας: Για όλα τα ζεύγη κόμβων u , υ που ανήκουν στο σύνολο των κορυφών V, θα πρέπει να ισχύει η εξής προϋπόθεση. 0<= f(u, υ) <= c(u, υ).

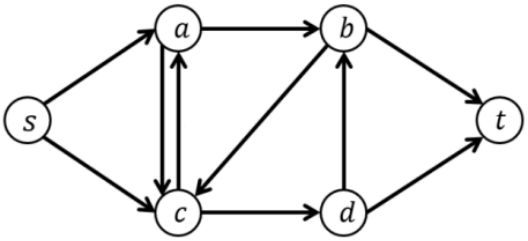
Διατήρηση ροής : Για όλους τους κόμβους u που ανήκουν στο σύνολο των κορυφών V εκτός του αφετηριακού κόμβου s και του τερματικού κόμβου να ισχύει ότι :

$\sum_{\upsilon \in V} f(\upsilon, u) = \sum_{\upsilon \in V} f(u, \upsilon)$

Όταν (u, υ) δεν ανήκει στο σύνολο των ακμών E, δεν μπορεί και να υπάρξει ροή από τον u μέχρι τον υ, και επομένως το f(u, υ)=0.

Η μη αρνητική ποσότητα f(u, υ) ονομάζεται ροή από τον κόμβο u προς τον κόμβο υ. Η τιμή |f| μια ροής ισούται με $|f| = \sum_{\upsilon \in V} f(s, \upsilon) - \sum_{\upsilon \in V} f(\upsilon, s)$.

Δηλαδή με την συνολική εκροή από τον αφετηριακό κόμβο μείον την εισροή στον αφετηριακό κόμβο υπολογίζουμε την τιμή ροής . Με αφετηρία τη θέση αυτή ένα δίκτυο ροής δεν έχει εισερχόμενες ακμές στον αφετηριακό κόμβο, οπότε η εισροή στον κόμβο αυτό, που εκφράζεται από το άθροισμα $\sum_{\upsilon \in V} f(\upsilon, s)$, είναι μηδέν. Ο λόγος που εισάγουμε τον όρο αυτό είναι ότι η εισροή στον αφετηριακό κόμβο έχει σημασία για τα υπολειπόμενα δίκτυα, τα οποία θα περιγράψουμε παρακάτω.



Σχήμα 1.5 Ενδεικτικό δίκτυο ροής

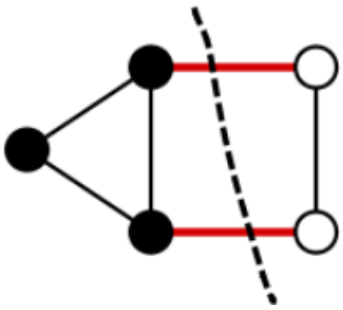
1.6 Γραφήματα και Τομές

Στην υποενότητα ορίζουμε και περιγράφουμε έναν θεωρητικό ορισμό της τομής ενός γραφήματος. Επιπρόσθετα γίνεται μια εισαγωγή σε ορισμένους χρήσιμους συμβολισμούς.

Ορισμός 1.8 Έστω ένα γράφημα G=(V, E), μη κατευθυνόμενο γράφημα με βάρη στις ακμές και με σύνολο κόμβων V και σύνολο ακμών E. Ως τομή ενός γραφήματος ορίζουμε την διαμέριση των κόμβων σε δύο μη κενά υποσύνολα A και B όπου B=V-A. Το βάρος της τομής του γραφήματος ορίζεται ως το βάρος των ακμών που περνάνε από το σύνολο A στο σύνολο B. Στο σχήμα 1.6 φαίνεται μια τομή που χωρίζει το γράφημα σε δύο σύνολα A και B όπου το σύνολο A χαρακτηρίζεται από τις μαύρες κορυφές και αντίστοιχα το σύνολο B από τις άσπρες κορυφές.

Τομή: Διαμέριση του συνόλου των κόμβων σε δύο μη κενά σύνολα (A U B= V, A ∩ B=∅)

Βάρος τομής: $w(A,B)= \sum_{u\in A\, v\in B} w(u,v)$



Σχήμα 1.6 Ενδεικτική τομή ενός γραφήματος

Κεφάλαιο 2. Ελάχιστη τομή με ροή δικτύου

2.1 Ορισμός του προβλήματος

Ένα από τα πιο σημαντικά προβλήματα στην θεωρία γραφημάτων είναι το πρόβλημα της ελάχιστης τομής *(Minimum Cut)*. Ιδιαίτερη σημασία έχει να αναφέρουμε ότι η κλάση προβλημάτων της ελάχιστης τομής έχει πολλές εφαρμογές σε πολλούς κλάδους της πληροφορικής όπως στην ψηφιακή επεξεργασία εικόνας *(digital image processing)*, στην ομαδοποίηση δεδομένων *(data aggregation),* μέχρι και στην εξόρυξη δεδομένων *(data mining).* Το συγκεκριμένο πρόβλημα ανάγεται σε ένα πρόβλημα διαμέρισης κόμβων ενός μη κατευθυνόμενου γραφήματος G (V,E) με βάρη ακμών, και χωρίζεται σε δύο υποσύνολα έτσι ώστε το άθροισμα των βαρών των ακμών που έχουν το ένα άκρο τους στο ένα υποσύνολο και το άλλο άκρο τους στο άλλο υποσύνολο, να ελαχιστοποιείται. Μέσα από αυτό το πρίσμα εμφανίζεται και η στενή σχέση τους προβλήματος της ελάχιστης τομής με την ροή δικτύου και πιο συγκεκριμένα με το πρόβλημα της μέγιστης ροής που θα αναλύσουμε παρακάτω. Οι περισσότεροι αλγόριθμοι που έχουν αναλύσει το πρόβλημα της μέγιστης ροής έχουν αποδείξει μια στενή σχέση με το πρόβλημα της ελάχιστης τομής και αυτό θα το αποδείξουμε και εμείς κατά την διάρκεια της διατριβής μας. Η κατηγορία αλγορίθμων που θα αναλύσουμε, επιλύει το πρόβλημα της εύρεσης ελάχιστης τομής βασιζόμενη στην μέθοδο των Ford Fulkerson από την οποία γνωρίζουμε ότι η τιμή της μέγιστης (s,t) ροής *(maximum (s,t) flow)* είναι ίση με την τιμή της ελάχιστης τομής *(min (s,t) cut),* έχοντας σταθερό τον αρχικό κόμβο s και κάθε φορά διαφορετικό τερματικό κόμβο t. Το συγκεκριμένο θεώρημα μας δίνει την δυνατότητα να αποδείξουμε ότι αποτελεί τη βάση όλων των αλγορίθμων που εντάσσονται στην κατηγορία αυτή. Επίσης θα πρέπει να αναφέρουμε ότι η συγκεκριμένη κατηγορία δεν υπολογίζει μόνο τις ελάχιστες τομές σε μη κατευθυνόμενα γραφήματα αλλά μας δίνει την δυνατότητα να υπολογίζει την ελάχιστη τομή και σε κατευθυνόμενα γραφήματα, αντίθετα με μια άλλη κατηγορία, που θα δούμε παρακάτω οπού υπολογίζει την ελάχιστη τομή μόνο για μη κατευθυνόμενα γραφήματα.

2.2 Το πρόβλημα της μέγιστης ροής

Στην παρούσα υποενότητα θα αναφερθούμε σε βασικές ιδιότητες και εργαλεία που αφορούν στις ροές. Με βάση αυτές τις ιδιότητες θα αναφερθούμε στον πιο αναγνωρισμένο αλγόριθμο στο υπολογισμό της μέγιστης ροής, τον αλγόριθμο Ford Fulkerson ενώ παράλληλα θα αναφέρουμε ίσως το πιο σημαντικό θεώρημα σε αυτήν την κλάση προβλημάτων, το θεώρημα μέγιστης ροής – ελάχιστής τομής το οποίο με τη σειρά του αποδεικνύει την ορθότητα της οικογένειας αλγορίθμων για εύρεση μέγιστης ροής, οι οποίοι βασίζεται στην επαύξηση μονοπατιού.

Το πρόβλημα της μέγιστης ροής το συναντάμε σε πολλές εφαρμογές κυρίως ως υποπρόβλημα σε άλλες εφαρμογές όπως πχ σε δίκτυα μεταφοράς, δίκτυο ηλεκτρικού φορτίου, κλπ... Με δεδομένο ότι έχουμε αναφέρει στην προηγούμενη ενότητα τον ορισμό ενός δικτύου ροής, ένας φυσικός στόχος είναι διευθέτηση της κίνησης έτσι ώστε να γίνεται όσο το δυνατόν πιο αποδοτική χρήση της διαθέσιμης χωρητικότητας. Έτσι το βασικό αλγοριθμικό πρόβλημα που θα εξετάσουμε είναι το ακόλουθο: Με δεδομένο ένα δίκτυο ροής, πρέπει να βρεθεί μια ροή με τη μέγιστη δυνατή τιμή. Καθώς θα μελετάμε το σχεδιασμό αλγορίθμων για αυτό το πρόβλημα, είναι χρήσιμο να εξετάσουμε με ποιον τρόπο η δομή του δικτύου ροής θέτει άνω όρια ως προς την μέγιστη τιμή μιας ροής s-t . Δεν πρέπει ωστόσο να αγνοηθεί ένα βασικό εμπόδιο ως προς την ύπαρξη μεγάλων ροών όπου είναι το εξής: Έστω ότι υποθέσουμε ότι διαιρούμε τις κορυφές του γραφήματος σε σύνολα A και B όπου $s \in A$ και $t \in B$. Έτσι λοιπόν με βάση την αίσθηση, οποιαδήποτε ροή από το s προς το t θα πρέπει σε κάποιο σημείο να περνάει από το σύνολο A και το σύνολο B, άρα θα πρέπει να χρησιμοποιεί κάποιες από τις χωρητικότητες ακμών από το A προς το B. Εύλογα προκύπτει το συμπέρασμα ότι οποιαδήποτε τέτοια διαίρεση του γραφήματος θέτει ένα όριο ως προς την μέγιστη δυνατή τιμή της ροής. Όπως αναφέρθηκε και παραπάνω ο αλγόριθμος Ford-Fulkerson που θα αναπτύξουμε παρακάτω είναι συνυφασμένος με την απόδειξη ότι η τιμής της μέγιστης ροής είναι ίση με την ελάχιστη χωρητικότητα οποιασδήποτε τέτοιας διαίρεση, που ονομάζεται ελάχιστη τομή.

2.3 Αλγόριθμος των Ford Fulkerson

Στην παρούσα υποενότητα παρουσιάζουμε τον αλγόριθμο υπολογισμού της μέγιστης ροής. Το 1956 οι μαθηματικοί Ford και Fulkerson παρουσίασαν και διατύπωσαν έναν αλγόριθμο για τον υπολογισμό των μέγιστων ροών που θεωρείται πλέον κλασσικός στην θεωρία γραφημάτων. Ο αλγόριθμος σχετίζεται με τρεις βασικές τεχνικές, οι οποίες

υπερβαίνουν κατά πολύ τα στενά όρια του ίδιου του αλγόριθμου και σχετίζονται με πολλούς αλγορίθμους ροής και με πολλά παρόμοια προβλήματα όπως τα υπολειπόμενα δίκτυα, τις επαυξητικές διαδρομές, και τις τομές. Οι έννοιες αυτές που αναφέραμε είναι θεμελιώδους σημασίας για το θεώρημα μέγιστης ροής – ελάχιστης τομής.

Ας υποθέσουμε ότι θέλουμε να βρούμε την μεγίστη ροή σε ένα δίκτυο, ξεκινάμε με μηδενική ροή που την ορίζουμε με f(e)=0 για όλα τα e. Είναι προφανές ότι αυτό είναι συνεπές με τις συνθήκες χωρητικότητας και διατήρησης. Θα προσπαθήσουμε λοιπόν να αυξήσουμε την τιμή του προωθώντας ροή κατά μήκος μιας διαδρομής που αποτελείται από κάποιες συγκεκριμένες ακμές εξακολουθώντας πάντα να μην παραβιάζει τις συνθήκες χωρητικότητα και διατήρησης. Αυτός είναι ένας γενικότερος τρόπος για την για την προώθηση ροής. Μπορούμε να προωθήσουμε προς τα εμπρός σε ακμές που έχει περισσέψει η χωρητικότητα αλλά και προς τα πίσω σε ακμές που μεταφέρουν ήδη ροή, έτσι ώστε να το εκτρέψει σε διαφορετική κατεύθυνση. Σε αυτό το σημείο μπορούμε να ορίσουμε τώρα το υπολειπόμενο δίκτυο *(residual network),* το οποίο παρέχει ένα συστηματικό τρόπο για την αναζήτηση ευθύδρομων και ανάδρομων λειτουργιών.

Υπολειπόμενο δίκτυο : Με δεδομένο ένα δίκτυο ροής G και μια ροή f στο G, ορίζουμε το υπολειπόμενο δίκτυο G_f του G ως προς τη ροή f με τον εξής τρόπο.

1. Το σύνολο των κορυφών του G_f είναι ίδιο με εκείνο του G.
2. Για κάθε ακμή e=(u, v) του G στην οποία $f(e)<c_e$, υπάρχει c_e-f(e) υπολειπόμενη χωρητικότητα για την οποία θα μπορούσαμε να προσπαθήσουμε να προωθηθεί ροή προς τα εμπρός. Με αυτό το τρόπο συμπεριλαμβάνουμε την ακμή e=(u, v) στο G_f με χωρητικότητα c_e-f(e).
3. Για κάθε ακμή e=(u, v) του G στην οποία f(e)>0, υπάρχουν f(e) μονάδες ροής τις οποίες μπορούμε να αναστρέψουμε την ροή προς τα πίσω. Έτσι με αυτό τρόπο συμπεριλαμβάνουμε την ακμή e'=(v,u) στο G_f με χωρητικότητα στο f(e). Κρίνεται σκόπιμο να αναφερθεί ότι η ακμή e' έχει ίδια άκρα με την e αλλά με αντίστροφη διεύθυνση. Αυτές τις ακμές τις αποκαλούμε ανάδρομές *(backward edges).*

Με αυτό το τρόπο ολοκληρώνεται ο ορισμός για το υπολειπόμενο δίκτυο G_f. Θα πρέπει να σημειωθεί ότι η κάθε ακμή e του δικτύου G μπορεί να δημιουργήσει μια ή δύο ακμές στο δίκτυο G_f. Αν $0< f(e)<c_e$ τότε θα συμπεριληφθεί στο G_f και μια ευθύδρομη ακμή και μια ανάδρομη ακμή. Άρα το G_f έχει το πολύ διπλάσιες ακμές από το G.

Επαυξητικές διαδρομές : Προχωρώντας πιο πέρα διαπιστώνει κανείς με ακρίβεια τον τρόπο όπου προωθούμε την ροή από την αρχική κορυφή s προς την τερματική κορυφή t

στο δίκτυο G_f. Έστω ότι ορίζουμε μια απλή διαδρομή s-t στο G_f με τον συμβολισμό P και με την ιδιότητα ότι η P δεν επισκέπτεται καμία κορυφή παραπάνω από μια φορά. Στηριζόμενοι στο δεδομένο μπορούμε να ορίσουμε ότι η bottleneck(P, f) είναι η ελάχιστη υπολειπόμενη χωρητικότητα σε οποιασδήποτε ακμή της P, σε σχέση με την ροή f.

Κάπως έτσι δημιουργείται η ανάγκη να ορίσουμε την ακόλουθη λειτουργία augment(f, P), η οποία παράγει μια νέα ροή f ′ στο G.

```
Augment (f, P)
        Έστω ότι b=bottleneck(P, f)
        For όλες τις ακμές (u, v) ∈ P
                IF e= (u, v) είναι ευθύδρομη ακμή τότε
                        Αύξηση το f(e) στο G κατά b
                Else (το (u, v) είναι ανάδρομη ακμή, και έστω e=(v, u))
                        Μείωσε το f(e) στο G κατά b
                Endif
        Endfor
        Return(f)
```

Ψευδοκώδικας Augment

Η εκτέλεση αυτής της λειτουργίας ήταν ένας σημαντικός λόγος για τον ορισμό του υπολειπόμενου γραφήματος. Πολλοί συχνά στην βιβλιογραφία αναφέρονται και σε οποιαδήποτε διαδρομή s-t στο υπολειπόμενο γράφημα ως διαδρομή επαύξησης *(augment path).* Εύκολα λοιπόν προκύπτει το συμπέρασμα το συμπέρασμα ότι το αποτέλεσμα της λειτουργίας augment(f, P) είναι μια νέα ροή f ′ στο G, η οποία λαμβάνεται με αύξηση και μείωση των τιμών ροής στις ακμές της διαδρομής P.

Αυτή η λειτουργία που αναφέραμε αποτυπώνει την διαδικασία ευθύδρομης και ανάδρομης προώθηση της ροής που αναφέραμε προηγουμένως. Ας εξετάσουμε τώρα τον παρακάτω αλγόριθμο με την μορφή ψευδοκώδικα για τον υπολογισμό μια ροής s-t στο δίκτυο G.

Αλγόριθμος Ford Fulkerson (G, s, t, c)

Input: Δίκτυο G (V, E) με χωρητικότητες στις ακμές { c_e : e ∈ E}.

Output: Μέγιστη Ροή

Αρχικά f(e)=0 για όλα τα : e ∈ E

Ενόσω υπάρχει μια επαυξητική διαδρομή s-t στο υπολειπόμενο δίκτυο G_f

Έστω ότι η P είναι μια απλή διαδρομή s-t στο G_f

f'=augment (P, f)

Ενημέρωσε το f σε f'

Ενημέρωσε το υπολειπόμενο δίκτυο G_f σε $G_{f'}$

Τέλος επανάληψης

Επέστρεψε f

Αλγόριθμος Ford Fulkerson

Ο αλγόριθμος που περιγράφτηκε είναι κατανοητός και η υλοποίηση του είναι σχετικά εύκολη. Όσο αναφορά την πολυπλοκότητα του αλγορίθμου καταλήγουμε πως ο Ford-Fulkerson εκτελείται σε χρόνο O(mf) υποθέτοντας ότι οι χωρητικότητες των ακμών είναι μη αρνητικοί ακέραιοι αριθμοί, όπου m ο αριθμός των ακμών και f είναι η μέγιστη ροή του δικτύου. Εάν πραγματοποιηθεί μια αναποτελεσματική υλοποίηση ή μια κακή επιλογή στην αναζήτηση για μονοπάτια προσαύξησης στο G_f τότε ο αλγόριθμος Ford Fulkerson τρέχει στην χειρότερη περίπτωση O((m+V)f) αντί για O(mf) όπου V ο αριθμός κορυφών του γραφήματος.

2.3.1 Θεώρημα μέγιστης ροής – ελάχιστης τομής

Το συγκεκριμένο θεώρημα αποδεικνύει ότι η ροή που επιστρέφεται από τον αλγόριθμο του Ford-Fulkerson ισούται με την ελάχιστη τομή ενός γραφήματος. Παρακάτω παρουσιάζουμε μια απόδειξη αυτού του θεωρήματος.

Θεώρημα:

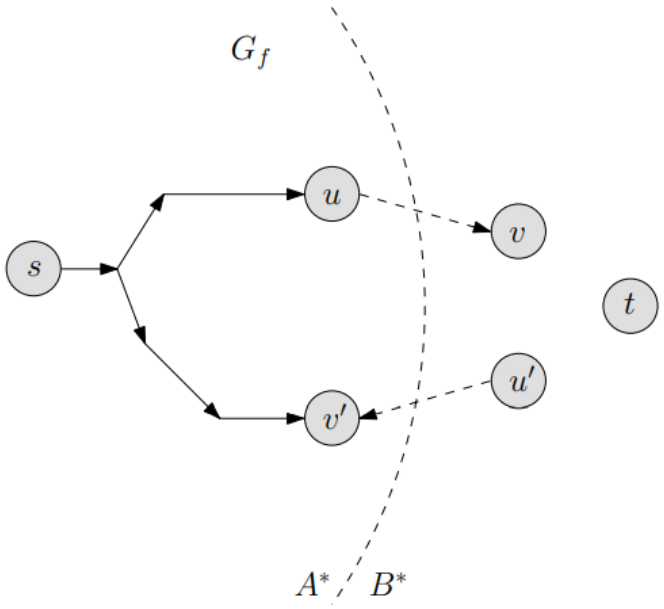
Έστω η f είναι μια ροή s-t τέτοια ώστε να μην υπάρχει διαδρομή s-t στο υπολειπόμενο γράφημα G_f. Τότε υπάρχει μια s-t τομή (A*, B*) στο G για την οποία v(f)=c(A*, B*) όπου v(f) ορίζουμε το συμβολισμό της τιμής μιας ροής f. Αυτό έχει ως αποτέλεσμα η f να έχει την μέγιστη τιμή από όλες της ροές του G και η (A*, B*) έχει την ελάχιστη χωρητικότητα από όλες τις τομές s-t του G.

Απόδειξη:

Έστω A* το σύνολο των κόμβων v ∈ G που υπάρχει διαδρομή s-v στο G_f και έστω B*= V\A*. Το (A*, B*) είναι μια s-t τομή αφού δεν υπάρχει διαδρομή s-t στο G_f και άρα το s ∈ A* ενώ το t ∈ B*.

- Έστω e=(u, v) ∈ G έτσι ώστε u ∈ A* και v ∈ B*. Τότε η ακμή είναι κορεσμένη, f(e)=c_e, αλλιώς η e θα ήταν ευθύδρομη στο G_f.
- Έστω e'=(u', v') ∈ G έτσι ώστε u' ∈ B* και v' ∈ A*. Τότε η ακμή είναι αχρησιμοποίητη, f(e)=0, αλλιώς η ανάδρομη της e''=(v', u') θα υπάρχει στο G_f.

Άρα v(f)=f_{out}(A*)-f_{in}(A*) = ∑f(e) - ∑f(e)= ∑c_e – 0 = c(A*,B*).



Σχήμα 2.1 Η τομή (A*, B*)

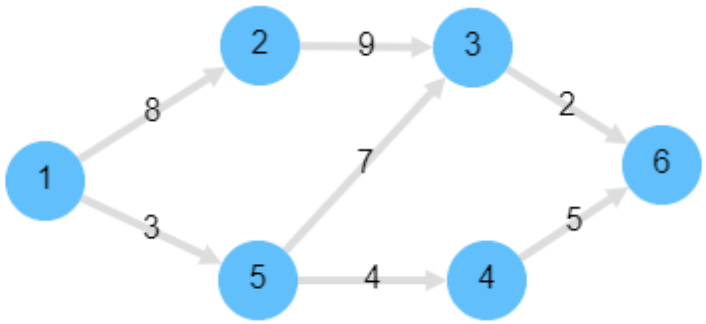
Συνοψίζοντας μπορούμε να διατυπώσουμε πιο γενικά το θεώρημα ως εξής: Σε όλα τα δίκτυα ροής η μέγιστη τιμή μιας ροής s-t είναι ίση με την ελάχιστη χωρητικότητα μιας τομής s-t.

2.4 Παράδειγμα των Ford Fulkerson

Η βασική ιδέα του αλγορίθμου Ford Fulkerson είναι επαναλαμβανόμενη εύρεση μιας επαυξητικής διαδρομής στο υπολειπόμενο δίκτυο, μέχρι να μην υπάρχει άλλο μονοπάτι από το s στο t. Στην παρούσα υποενότητα αφότου έχουμε αναλύσει και εμπεδώσει τον αλγόριθμο Ford Fulkerson παρουσιάζουμε ένα απλό και κατανοητό παράδειγμα έτσι

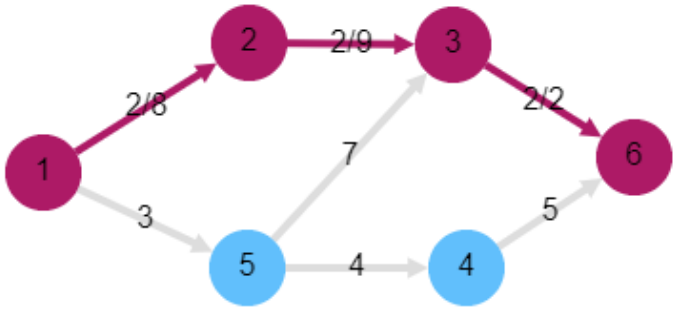
ώστε να είμαστε σε θέση να καταλάβουμε και την πρακτική υλοποίηση του αλγορίθμου πάνω σε ένα τυχαίο γράφημα G.

Ας υποθέσουμε λοιπόν ότι έχουμε ένα γράφημα G(V, E) που αποτυπώνεται παρακάτω, και ότι θέλουμε να βρούμε την μέγιστη ροή του. Ο κόμβος πηγής είναι ο S=1 και ο τερματικός κόμβος είναι ο T=6.

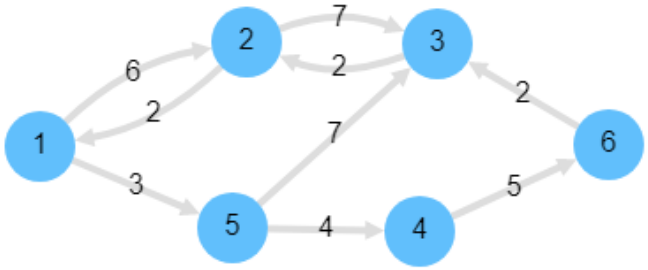


Σχήμα 2.2 Δίκτυο ροής G

Βήμα 1:
Επιπρόσθετα επιλέγουμε μια αυθαίρετη διαδρομή από S έως T. Σε αυτό το βήμα, επιλέγουμε την 1->2->3->6.



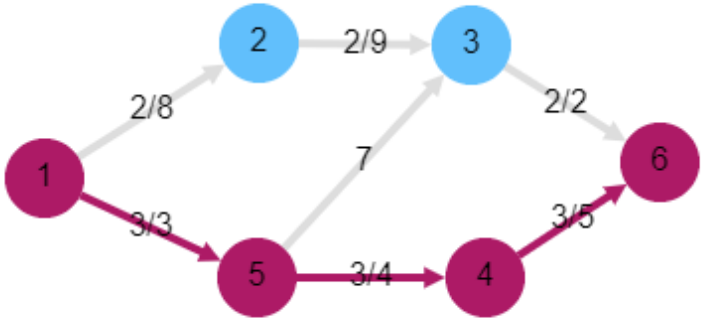
Σχήμα 2.3 Ενδεικτική ροή 1->2->3->6



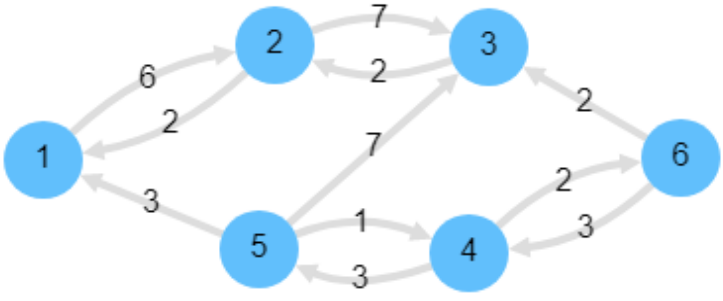
Σχήμα 2.4 Το υπολειπόμενο δίκτυο

Η ελάχιστη χωρητικότητα μεταξύ των τριών ακμών είναι 2. Και με βάση αυτό ενημερώνουμε την ροή/χωρητικότητα *(flow/capacity)* για κάθε διαδρομή.

Βήμα 2:
Σε αυτό το βήμα επιλέγουμε μια νέα αυθαίρετη διαδρομή από S έως T. Σε αυτό το βήμα, επιλέγουμε την 1->5->4->6.



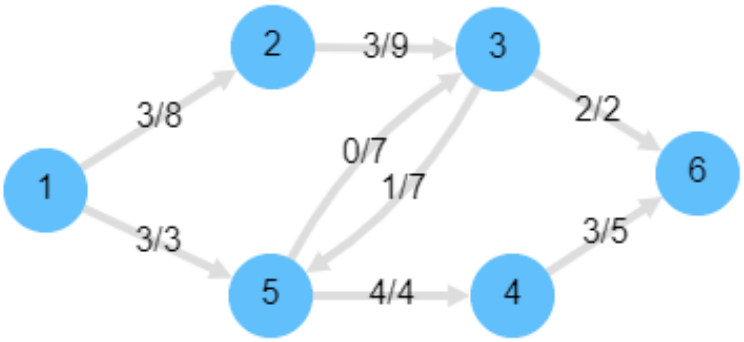
Σχήμα 2.5 Ενδεικτική ροή 1->5->4->6



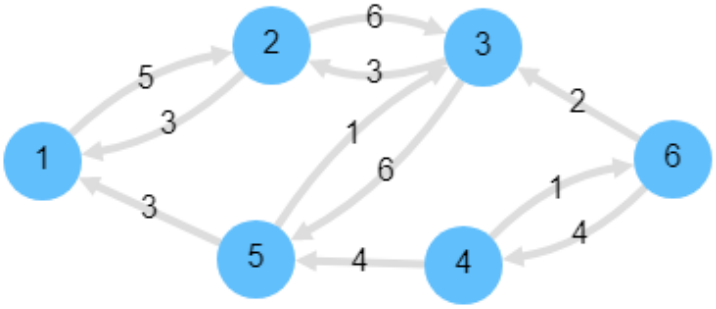
Σχήμα 2.6 Το υπολειπόμενο δίκτυο

Η ελάχιστη χωρητικότητα μεταξύ των τριών ακμών είναι 3. Και με βάση αυτό ενημερώνουμε την ροή/χωρητικότητα *(flow/capacity)* για κάθε διαδρομή.

Βήμα 3:
Σε αυτό το βήμα θα εξετάσουμε και την αντίστροφη διαδρομή βάση του αλγορίθμου, όπου είναι η 3->5 . Παράλληλα η ελάχιστη υπολειπόμενη χωρητικότητα της διαδρομής 1->2->3->5->4->6 των ακμών είναι λόγο των (3->5).



Σχήμα 2.7 Ενημέρωση χωρητικότητας δικτύου



Σχήμα 2.8 Το υπολειπόμενο δίκτυο

Βήμα 4:

Στο τέταρτο και τελευταίο βήμα αφού έχουμε βρει όλες τις ροές τις προσθέτουμε και έχουμε το αποτέλεσμα 2+3+1=6, όποτε η μέγιστη δυνατή ροή είναι 6 για το γράφημα G.

Εάν με κάποιο τρόπο θα έπρεπε να υπολογίζουμε την ελάχιστη τομή για ένα μη κατευθυνόμενο το όποιο θα ήταν παρόμοιο με το παραπάνω γράφημα θα έπρεπε να εκτελέσουμε τον αλγόριθμο τον Ford Fulkerson 5 φορές έχοντας κάθε φορά διαφορετικό τερματικό κόμβο.

Κεφάλαιο 3. Αλγόριθμος Stoer & Wagner

3.1 Αλγόριθμοι ελάχιστης τομής

Όπως είχαμε αναφέρει και στην προηγούμενη ενότητα η εύρεση της ελάχιστης τομής σε ένα μη κατευθυνόμενο γράφημα αποτελεί για την θεωρία γραφημάτων ένα κλασσικό θεμελιώδες αλγοριθμικό πρόβλημα. Εύκολα λοιπόν βγάζει κανείς το συμπέρασμα ότι το πρόβλημα της εύρεσης ελάχιστης τομής σε ένα μη κατευθυνόμενο γράφημα G με ακμικά βάρη έχει μελετηθεί εκτενώς και έχουν προταθεί πολλοί σχετικοί αλγόριθμοι. Η κατηγορία αλγορίθμων που θα αναλυθεί στην παρούσα ενότητα και πού ανήκει ο αλγόριθμος Stoer-Wagner, επιλύει το πρόβλημα της εύρεσης ελάχιστης τομής ενός γραφήματος χρησιμοποιώντας την επαναληπτική μέθοδο συρρίκνωσης ακμών μεγάλου βάρους οι οποίες δεν συμμετέχουν στην ελάχιστη τομή. Πρώτου αναλύσουμε όμως τον αλγόριθμο Stoer-Wagner θα αναφερθούμε συνοπτικά και σε κάποιους άλλους αλγόριθμους που ανήκουν στην κατηγορία που αναφέραμε.

3.1.1 Αλγόριθμος Nagamochi-Ibaraki

Το 1992 οι Nagamochi και Ibaraki παρουσίασαν έναν αλγόριθμο που υπολογίζει την ελάχιστη τομή χωρίς να καταφεύγει σε υπολογισμούς μέγιστης ροής. Συνοπτικά ο αλγόριθμος εκτελείται σε φάσεις και διατηρεί μια τιμή λ, η οποία αντιστοιχεί στην ελάχιστη χωρητικότητα τομής που έχει βρεθεί μέχρι την δεδομένη χρονική στιγμή. Σε κάθε φάση του αλγορίθμου αναγνωρίζει μια ακμή uw για την οποία ικανοποιεί τη σχέση $\lambda_{uw}(G) \geq \lambda$, όπου η $\lambda_{uw}(G)$ είναι μια συνάρτηση βάρους που εκχωρεί μη αρνητικά βάρη στις ακμές του γραφήματος G. Εάν βρεθεί μια τέτοια ακμή στο γράφημα γνωρίζουμε ότι αυτή ακμή μπορεί να συρρικνωθεί. Αύτη η διαδικασία επαναλαμβάνεται n-1 φόρες και βρίσκεται λύση για το πρόβλημα της ελάχιστης τομής. Παράλληλα χρησιμοποιείται μια ρουτίνα ContractSafe (που δεν θα αναλυθεί στην παρούσα διατριβή), η οποία συρρικνώνει την ακμή και επιστρέφει το συρρικνωμένο γράφημα G \ {u, w} (contracted graph) μαζί με τη νέα εκτίμηση της ελάχιστης τομής. Επαγωγικά λοιπόν σκεπτόμενοι καταλήγουμε στο συμπέρασμα ότι σε κάθε εκτέλεση του αλγόριθμου συρρικνώνεται μια

τουλάχιστον ακμή και αυτή είναι η τελευταία ακμή που ‘’σαρώνεται’’. Επομένως η πολυπλοκότητα του αλγόριθμου είναι O(mn+(n^2)logn). Παρακάτω περιγράφεται ο αλγόριθμος και σε μια τυπική μορφή ψευδοκώδικα.

Αλγόριθμος NI(G(V, E))

λ=min{ c(u) : u ∈ *V* }

(ο παραπάνω συμβολισμός σημαίνει ότι ελάχιστη χωρητικότητα τομής ισούται με το ελάχιστο άθροισμα των βαρών που προσπίπτουν στην κορυφή u)

Ενόσω (|V|≥3)

(G, λ) <- ContractSafe (G, λ)

Επίστρεψε λ

Αλγόριθμος Nagamochi-Ibaraki

3.1.2 Αλγόριθμος Karger-Stein

Το 1996 οι Karger Stein παρουσίασαν έναν αλγόριθμο που είναι σε μεγάλο βαθμό παρόμοιος με των Nagamochi-Ibaraki όπου επαναληπτικά συρρικνώνει ακμές. Αντίθετα όμως με τον παραπάνω αλγόριθμο ο Karger Stein έχει την δυνατότητα κάποιες φορές να συρρικνώσει ακμή η οποία να ανήκει στην ελάχιστη τομή. Η βασική ιδέα του αλγορίθμου είναι η εκτέλεση μιας σειράς συρρίκνωσης ακμών στο γράφημα, η οποία μειώνει τον αριθμό των κορυφών, μέχρι να μείνουν μόνο δυο κορυφές. Η ελάχιστη τομή προκύπτει από τον αριθμό ακμών που συνδέουν τις δύο εναπομείναντες κορυφές. Ο επαναληπτικός αλγόριθμος συρρίκνωσης Karger-Stein εκτελείται σε χρόνο O((n^2)logn). Παρακάτω περιγράφεται ο αλγόριθμος και σε μια τυπική μορφή ψευδοκώδικα.

INPUT: (Graph G = (V, E), t):

OUTPUT: Η ελάχιστη τομή του γραφήματος

Επανέλαβε την ακόλουθη διαδικασία t φορές:

Καθώς |V|>t:

Επέλεξε μια τυχαία ακμή e από το σύνολο E

G<-G/e

(Στο παραπάνω βήμα, διαγράφεται η ακμή e από το γράφημα G και συρρικνώνονται οι δύο κόμβοι u και v σε έναν νέο κόμβο, ο οποίος αντικαθιστά τους δύο προηγούμενους

κόμβους. Ο νέος κόμβος έχει όλες τις ακμές που ενώνουν τους προηγούμενους κόμβους, εκτός από την ακμή που διαγράφτηκε).

Επέστρεψε G

Αλγόριθμος Karger-Stein

3.2 Ορισμός και ανάλυση του αλγόριθμου

Εισαγωγή. Το 1997 ο Mechthild Stoer και ο Frank Wagner παρουσίασαν στην διατριβή τους έναν αλγόριθμο που έβρισκε την ελάχιστη τομή σε ένα μη κατευθυνόμενο γράφημα με ακμικά βάρη. Ο αλγόριθμος αυτός θεωρήθηκε απλός και η εφαρμογή του είναι σχετικά εύκολη. Ο συγκεκριμένος αλγόριθμος θεωρήθηκε καινοτόμος για το 1997 διότι σε αντίθεση με άλλους αλγορίθμους εύρεσης ελάχιστης τομής που είχαν βρεθεί πριν το 1997, ο αλγόριθμος αυτός δεν χρησιμοποιεί τεχνικές ροής. Ο αλγόριθμος αποτελείται από περίπου |V|-1 βήματα , όπου το κάθε βήμα αντιστοιχεί στην αναζήτηση της μέγιστης γειτνίασης των κορυφών *(maximum adjacency).* Θεωρούμε ότι έχουμε ένα απλό μη κατευθυνόμενο γράφημα G με σύνολο κορυφών V και με σύνολο ακμών E. Κάθε ακμή την ορίζουμε με το συμβολισμό e και ταυτόχρονα έχει μη αρνητικό βάρος w(e). Προσπαθώντας να προσεγγίσουμε τον αλγόριθμο μπορούμε να παρατηρήσουμε ότι , αν γνωρίζουμε πώς να βρεθούν δύο κορυφές s και t και το βάρος της ελάχιστης (s, t)- τομής, τότε έχουμε βρει την λύση του προβλήματος με την βοήθεια του παρακάτω θεωρήματος.

Θεώρημα 3.1 Έστω s και t δύο κορυφές ενός γραφήματος G. Έστω G\{s,t} το γράφημα που προκύπτει από την συνένωση των κορυφών s και t . Τότε η ελάχιστη τομή του G μπορεί να προκύψει παίρνοντας το μικρότερο από τις ποσότητες ελάχιστης (s, t)-τομής του G και της ελάχιστης τομής του G\{s,t} .

Η απόδειξή αυτού του θεωρήματος ανάγεται σε μια απλή συλλογιστική πορεία. Έστω ότι η ελάχιστη τομή του G χωρίζει τις κορυφές s, t τότε η ελάχιστη (s, t)- τομή του G είναι και η ελάχιστη τομή του G. Σε αντίθετη περίπτωση η ελάχιστη τομή του G είναι η ελάχιστη τομή του G\{s,t}. Ως συνέπεια, δημιουργείται μια διαδικασία που υπολογίζει αυθαίρετα μια ελάχιστη τομή (s, t)- τομή του G. Αύτη χρησιμοποιείται επαναληπτικά έως ότου υπολογισθεί η ελάχιστη τομή του G. Συγκεκριμένη διαδικασία χρησιμοποιείται επαναληπτικά μέχρι να υπολογισθεί η ελάχιστη τομή του G και είναι γνωστή και στην βιβλιογραφία ως αναζήτηση της μέγιστης γειτνίασης *(maximum adjacency search or maximum cardinality search).* Παρακάτω παρουσιάζουμε ενδελεχώς την διαδικασία.

MinimumCutPhase (G, w, α)

A<-{α}

Καθώς A≠V

Πρόσθεσε στο σύνολο A την κορυφή που έχει την μεγαλύτερη συνδεσιμότητα με το σύνολο A και αποθήκευσε το cut-of-the-phase(δηλαδή την τομής της συγκεκριμένης φάσης), συρρίκνωσε το γράφημα G μέσω της συνένωσης των δύο τελευταίων κορυφών που προστέθηκαν στο A.

Αλγόριθμος MinimumCutPhase

Προχωρώντας πιο πέρα διαπιστώνει κανείς ότι σε κάθε φάση του αλγόριθμου δημιουργείται μια διάταξη γειτνίασης κορυφών A. Αυτό σημαίνει ότι το σύνολο A περιέχει μια τυχαία κορυφή και επαναληπτικά προστίθεται σε αυτό το σύνολο μια καινούργια κορυφή μέχρι όλες οι κορυφές του γραφήματος G εισαχθούν σε αυτό. Σε κάθε επανάληψη εισέρχεται μια κορυφή εκτός του συνόλου A που είναι περισσότερο συνδεδεμένη με το A. Με αλλά λόγια, εισέρχεται η κορυφή z για την οποία ισχύει

$$z \notin A \text{ και } w(A, z) = \max \{w(A, y) \mid y \notin A\},$$

όπου το w(A, z) συμβολίζει το άθροισμα των βαρών όλων των ακμών ανάμεσα στο σύνολο A και της κορυφής z. Στο τέλος της κάθε φάσης οι δύο κορυφές που προστέθηκαν τελευταίοι στο σύνολο A ενώνονται, με άλλα λόγια, οι δύο τελευταίες κορυφές αντικαθιστούνται με μια νέα κορυφή, και οι ακμές από τις δύο αυτές κορυφές προς μια οποιοδήποτε άλλη κορυφή αντικαθίστανται με μια ακμή όπου το βάρος της προκύπτει από το άθροισμα των δύο επί μέρους ακμών. Έχει ιδιαίτερη σημασία να αναφέρουμε ότι αν το γράφημα G έχει βρόγχους θα πρέπει να απομακρυνθούν. Η τομή στο σύνολο των κορυφών V που χωρίζει την κορυφή που προστέθηκε τελευταία στο σύνολο A από το υπόλοιπο γράφημα καλείται ως cut-of-the-phase. Το μικρότερο cut-of-the-phase αποτελεί την ελάχιστη τομή του G και με αυτό το τρόπο λύνεται το πρόβλημα του αλγόριθμου.

MinimumCut (G, w, α)

Καθώς |V|>1

Τότε εκτελείται η MinimumCutPhase(G, w, α)

Αν το cut-of-the-phase είναι μικρότερο από το τρέχον cut-of-the-phase

Τότε θέσε το cut-of-the-phase με την ελάχιστη τομή του γραφήματος G

Αλγόριθμος MinimumCut

Ορθότητα 3.3 Για την απόδειξη της ορθότητα τους αλγόριθμου παρατίθεται το ακόλουθο λήμμα.

Λήμμα 3.4 Κάθε cut-of-the-phase είναι η ελάχιστη (s,t)-Τομή στο τρέχον γράφημα , όπου s και t οι κορυφές που προστέθηκαν τελευταίες στη διάταξη γειτνίασης.

Η απόδειξη του παραπάνω λήμματος ανάγεται στον ισχυρισμό ότι σε κάθε φάση οι κορυφές του γραφήματος ταξινομούνται γραμμικά, ξεκινώντας από την κορυφή α και καταλήγοντας στην κορυφή s και στην κορυφή t. Ορίζουμε μια τυχαία (s , t)- τομή C για το τρέχον γράφημα θα πρέπει να αποδείξουμε ότι είναι τουλάχιστον τόσο μεγάλη όσο και το cut-of-the-phase. Έστω λοιπόν μια ενεργή *(active)* κορυφή υ != α *(ενεργή είναι μια κορυφή όταν αυτή και η κορυφή που προστέθηκε πριν από αυτήν ανήκουν σε διαφορετικά μέρη στη τομή C).* Με τον συμβολισμό w(C) ορίζουμε το συνολικό βάρος της τομής C. Το σύνολο των κορυφών που προστέθηκαν πριν από το υ συμβολίζεται με $A_{υ,}$ η τομή του συνόλου $A_υ \cup \{υ\}$ συμβολίζεται με $C_υ$ και το βάρος της με $w(C_υ)$. Σύμφωνα με την επαγωγή στο σύνολο των ενεργών κορυφών ότι για κάθε ενεργή κορυφή υ ισχύει η παρακάτω σχέση:

$$w(A_υ, υ) \leq w(C_υ)$$

Πρέπει να τονιστεί ότι για την πρώτη ενεργή κορυφή, η ανισότητα ισχύει με ισότητα. Έστω ότι η ανισότητα ισχύει για κάθε ενεργή κορυφή που προστίθεται μέχρι και την ενεργή κορυφή υ, και έστω ότι η u είναι η επόμενη ενεργή κορυφή που προστίθεται, το ισχύει η παρακάτω σχέση:

$$w(A_u, u) = w(A_υ, u) + w(A_u \setminus A_υ, u) =: α$$

Καθώς η κορυφή υ επιλέχτηκε πριν από την κορυφή u εύλογα καταλήγουμε στο συμπέρασμα ότι ισχύει $w(A_υ, u) \leq w(A_υ, υ)$, γιατί σύμφωνα με τον αλγόριθμο σε κάθε επανάληψη επιλέγεται η κορυφή με την μεγαλύτερη συνδεσιμότητα με το σύνολο $A_υ$. Θα ήταν σημαντική παράλειψη να μην τονιστεί ότι με βάση την επαγωγική μέθοδο ισχύει συγκεκριμένη ανισότητα $w(A_υ, υ) \leq w(C_υ)$. Συμπληρωματικά καθώς η u είναι μια ενεργή κορυφή , οι ακμές ανάμεσα στο σύνολο $A_u \setminus A_υ$ και της κορυφής u ενώνουν διαφορετικά μέρη της τομής C. Συνεπώς λοιπόν επηρεάζουν μόνο την ποσότητα $w(C_u)$ και όχι την ποσότητα $w(C_υ)$. Κατά συνεπεία ισχύει η παρακάτω σχέση:

$$w(A_u, u) = w(C_υ) + w(A_u \setminus A_υ, u) \leq w(C_u)$$

Ως αποτέλεσμα αφού η κορυφή t είναι πάντα ενεργή όσο αναφορά την τομή C, ισχύει ότι $w(A_t, t) \leq w(C_t)$. Συνεπώς το cut-of-the-phase αποτελεί και την ελάχιστη (s, t)- τομή.

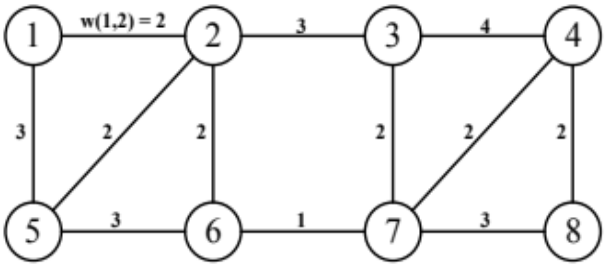
Χρόνος εκτέλεσης αλγορίθμου. Η πολυπλοκότητα χρόνου του αλγόριθμου MinimumCut είναι ουσιαστικά ίση με |V|-1 φορές την πολυπλοκότητα χρόνου της

διαδικασίας MinimumCutPhase, η οποία καλείται σε κάθε φάση σε γράφημα μειωμένου μεγέθους. Αυτό που πρέπει να δειχθεί είναι ότι μια εκτέλεση της MinimumCutPhase χρειάζεται το πολύ O(|E|+|V|log|V|) οδηγώντας σε μια συνολική πολυπλοκότητα O(|V||E|+(|V|^2)log|V|).

Το κλειδί για μια αποτελεσματική υλοποίηση της κάθε φάσης είναι το κατά πόσο εύκολα μπορεί να επιλεχτεί η επόμενη κορυφή που είναι η πιο ισχυρά συνδεδεμένη και που πρόκειται να προσδεθεί στο σύνολο A. Δηλαδή, η επόμενη κορυφή που εισάγεται είναι η κορυφή με την μεγαλύτερη γειτνίαση ως προς το σύνολο A. Κατά την διάρκεια της εκτέλεσης της φάσης, όλες οι κορυφές που δεν έχουν προσδεθεί στο σύνολο A, τοποθετούνται σε μια ουρά προτεραιότητας με βάση ένα πεδίο κλειδί. Το κλειδί μιας κορυφής u είναι το άθροισμα των βαρών των ακμών που συνδέουν την κορυφή αυτήν με το σύνολο A και συμβολίζεται ως w(A, u). Εύκολα γίνεται αντιληπτό ότι κάθε φορά που μια κορυφή u προστίθεται στο σύνολο A, η ουρά ανανεώνεται, με την κορυφή u να διαγράφεται από την ουρά και το κλειδί κάθε κορυφής w που δεν ανήκει στο A και συνδέεται με την κορυφή u πρέπει να μειωθεί κατά το βάρος της ακμής uw, εάν υπάρχει. Καθώς αυτό γίνεται ακριβώς μια φορά για κάθε ακμή, γενικά θα πρέπει να αποδώσουμε τις λειτουργίες |V| ExtractMax και |E| IncreaseKey. Χρησιμοποιώντας το σωρό Fibonacci [Fredman and Tarjun 1987], μπορούμε να εκτελέσουμε μια λειτουργία ExtractMax σε O(log|V|) χρόνο και μια λειτουργία IncreaseKey σε O(1) χρόνο. Έτσι λοιπόν ο συνολικός χρόνος που χρειαζόμαστε για αυτό το βασικό βήμα που είναι σημαντικό για την υπόλοιπη φάση, είναι O(|E|+|V|log|V|).

3.3 Παράδειγμα του αλγόριθμου Stoer Wagner

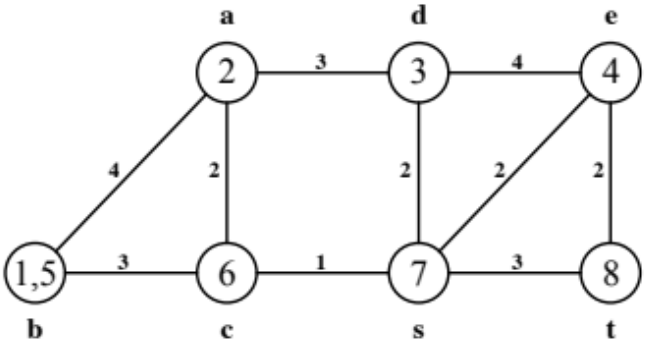
Στην παρούσα υποενότητα θα παρουσιάσουμε ένα παράδειγμα του αλγόριθμου Stoer Wagner σε ένα γράφημα για να δούμε και την πρακτική εφαρμογή του. Για τον υπολογισμό της ελάχιστης τομής του μη - κατευθυνόμενου γραφήματος G που φαίνεται στο σχήμα 3.1 ακολουθούνται τα παρακάτω βήματα.



Σχήμα 3.1 γράφημα G

Βήμα 1:

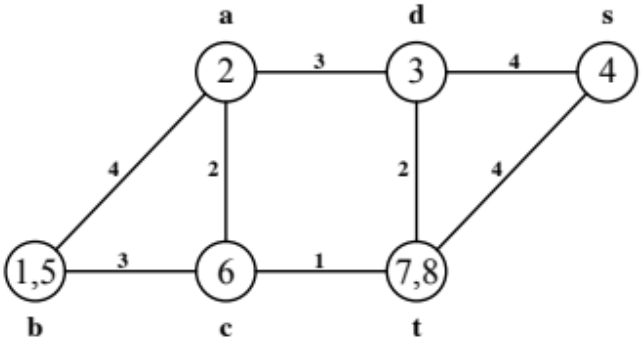
Πρώτα εκτελείται ο αλγόριθμος MinimumCutPhase που αναφέρθηκε πιο πάνω με είσοδο το γράφημα G και μια αρχική κορυφή 2 (όπως φαίνεται και στο σχήμα). Στην συνέχεια δημιουργείται μια διάταξη γειτνίασης Α η οποία αρχικά περιέχει την κορυφή 2. Υπολογίζοντας τις γειτνιάσεις των κορυφών του V(G)-2 όπου εισάγονται σε μια ούρα προτεραιότητας με κλειδί τη γειτνίαση της κάθε κορυφής του συνόλου αυτού με την κορυφή 2 (στην γενική περίπτωση το κλειδί να είναι η γειτνίαση της κορυφής με όλες τις κορυφές που έχουν ήδη εισαχθεί στην διάταξη Α). Στην συνέχεια επιλέγεται η κορυφή με την μέγιστη συνδεσιμότητα από την ουρά προτεραιότητας στο παράδειγμα μας είναι η κορυφή 3 και εισάγεται στο τέλος της διάταξης Α αφού πρώτα όμως έχει διαγραφτεί από την ουρά προτεραιότητας. Μετέπειτα για κάθε κορυφή u, στην ουρά προτεραιότητας, γείτονα του 3, αυξάνεται το κλειδί του κατά μια ποσότητα ίση με το βάρος της ακμής που συνδέει την κορυφή u με την κορυφή 3. Στην συνέχεια επιλέγεται η κορυφή 4, από την ουρά προτεραιότητας, και εισάγεται στο τέλος της διάταξης Α. Τα κλειδιά των κορυφών που είναι και γείτονες στην κορυφή 4, αυξάνονται με τον ίδιο τρόπο που περιγράφεται πιο πάνω . Η διαδικασία αυτή επαναλαμβάνεται έως ότου έχει διαγραφτεί και η τελευταία κορυφή από την ουρά προτεραιότητας. Κατά συνέπεια , προκύπτει η εξής διάταξη A={ 2,3,4,7,8,6,5,1}. Όλα αυτά μας οδηγούν στο συμπέρασμα ότι το κλειδί της τελευταίας κορυφής που έχει εισαχθεί στην διάταξη Α είναι ίσο με την ελάχιστη τομή των s-t όπου οι s, t είναι οι δύο τελευταίες κορυφές που μπήκανε στην διάταξη Α. Έχοντας σαν αφετηρία το θεώρημα 3.2.2 η ελάχιστη τομή του G είναι η ελάχιστη τομή του (s, t) ή είναι ελάχιστη τομή του γραφήματος G\{s, t} που προκύπτει μετά την μείωση των κορυφών s, t. Άρα κατά συνέπεια μια υποψήφια ελάχιστη τομή του γραφήματος G αποτελεί η ελάχιστη (s, t) τομή κατά την οποία το σύνολο των κορυφών του γραφήματος G χωρίζεται στα δύο υποσύνολα {1}, {2,3,4,5,6,7,8} όποτε η τιμή της τομής για τα δύο υποσύνολα είναι w=5. Όπου το w συμβολίζει το βάρος της τομής.



Σχήμα 3.2 το γράφημα G μετά την μείωση των κορυφών 1,5

Βήμα 2:

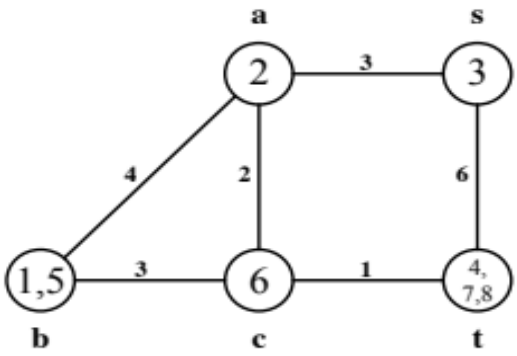
Στο μεταξύ η διαδικασία του βήματος 1 επαναλαμβάνεται για το γράφημα G\{s, t} και καταλήγει σε μια νέα διάταξη A={2,(1,5),6,3,4,7,8} και συνεπώς η ελάχιστη τομή του G αποτελείται από τα υποσύνολα {8}, {1,2,3,4,5,6,7} και η τιμή του βάρους της τομής είναι ίση με w=5 και επομένως οι τελευταίες κορυφές της διάταξης μειώνονται.



Σχήμα 3.3 το γράφημα G μετά την μείωση των κορυφών 7,8

Βήμα 3:

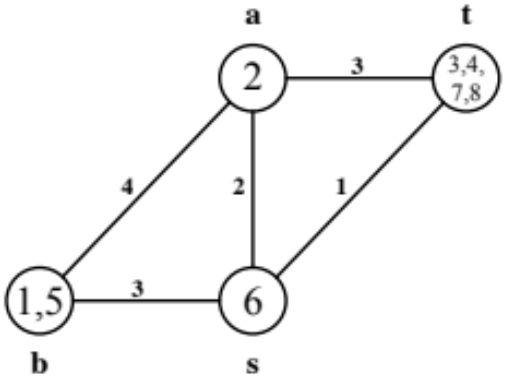
Η παραπάνω διαδικασία επαναλαμβάνεται για το γράφημα που προκύπτει στο σχήμα 3.3, και η υποψήφια ελάχιστη τομή του γραφήματος προκύπτει από τα δύο υποσύνολα {7, 8}, {1,2,3,4,5,6} και η τιμή του είναι ίση με w=7 και επομένως οι τελευταίες κορυφές της διάταξης μειώνονται.



Σχήμα 3.4 το γράφημα G μετά την μείωση των κορυφών 4,7,8

Βήμα 4:

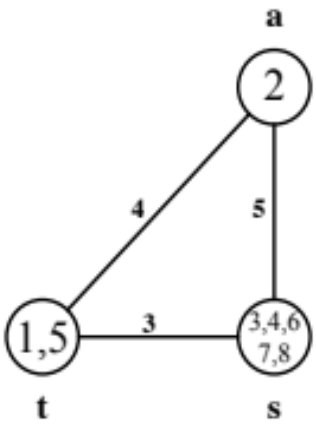
Η παραπάνω διαδικασία επαναλαμβάνεται για το γράφημα που προκύπτει στο σχήμα 3.4, και η υποψήφια ελάχιστη τομή του γραφήματος προκύπτει από τα δύο υποσύνολα {4,7,8}, {1,2,3,5,6} και η τιμή του είναι ίση με w=7 και επομένως οι τελευταίες κορυφές της διάταξης μειώνονται.



Σχήμα 3.5 το γράφημα G μετά την μείωση των κορυφών 3,4,7,8

Βήμα 5:

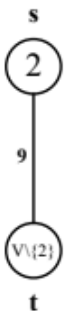
Η παραπάνω διαδικασία επαναλαμβάνεται για το γράφημα που προκύπτει στο σχήμα 3.5, και η υποψήφια ελάχιστη τομή του γραφήματος προκύπτει από τα δύο υποσύνολα {3,4,7,8}, {1,2,5,6} και η τιμή του είναι ίση με w=4 και επομένως οι τελευταίες κορυφές της διάταξης μειώνονται.



Σχήμα 3.6 το γράφημα G μετά την μείωση των κορυφών 3,4,6,7,8,

Βήμα 6:

Η παραπάνω διαδικασία επαναλαμβάνεται για το γράφημα που προκύπτει στο σχήμα 3.5, και η υποψήφια ελάχιστη τομή του γραφήματος προκύπτει από τα δύο υποσύνολα {2,3,4,6,7,8}, {1,5} και η τιμή του είναι ίση με w=7 και επομένως οι τελευταίες κορυφές της διάταξης μειώνονται.



Σχήμα 3.7 το γράφημα G μετά την τελευταία μείωση των κορυφών

Βήμα 7:

Στην τελευταία επανάληψη τους αλγορίθμου καταλήγουμε στην υποψήφια τομή που αντιστοιχεί στην διαμέριση {2}, V(G)-{2} και έχει τιμή ίση με w=9.

Συνοψίζοντας λοιπόν για κάθε MinCutPhase έχουμε το αντίστοιχο βάρος τομής.

Cut-of-the-phase	w
{1}, {2,3,4,5,6,7,8}	5
{8}, {1,2,3,4,5,6,7}	5
{7,8}, {1,2,3,4,5,6}	7
{4,7,8}, {1,2,3,5,6}	7
{3,4,7,8}, {1,2,5,6}	4
{1,5}, {2,3,4,6,7,8}	7
{2}, {1,3,4,5,6,7,8}	9

Επομένως καταλήγουμε στο συμπέρασμα ότι από τις n-1 υποψήφιες ελάχιστες τομές που υπολογίστηκαν σε κάθε βήμα του αλγορίθμου επιλέγουμε την ελάχιστη τομή η οποία έχει το μικρότερο βάρος τομής. Στο παράδειγμά μας αντιστοιχεί στο βήμα 4 όπου τα υποσύνολα που προέκυψαν είναι τα {3,4,7,8} και {1,2,5,6} όπου και το βάρος της τομής είναι w=4.

Κεφάλαιο 4. Περιγραφή Υλοποιήσεων

Στην ενότητα αυτή παρουσιάζουμε τις υλοποιήσεις των αλγορίθμων, με βάση την μελέτη που παρουσιάστηκε στις προηγούμενες ενότητες. Αρχικά παρουσιάζεται την πλατφόρμα και τα προγραμματιστικά συστατικά που χρησιμοποιήθηκαν, στην συνέχεια δίνονται οι λεπτομέρειες υλοποίησης για τους βασικούς αλγορίθμους του συστήματος καθώς και την δομή του κώδικα.

4.1 Περιγραφή Πακέτων

Στην υποενότητα αυτήν δίνεται μια λίστα από τα πακέτα (*package)* ή τις βιβλιοθήκες (θεωρείται ίδιος ορός στην επιστήμη των υπολογιστών) που υλοποιήθηκαν για την υλοποίηση των αλγορίθμων και ακολούθως θα τα αναλύσουμε περαιτέρω.

Πακέτο NetworkX:

Το NetworkX είναι μια βιβλιοθήκη της Python με ελεύθερο λογισμικό που ασχολείται με την μελέτη γραφημάτων και δικτυών, η βιβλιοθήκη μας δίνει πολλές δυνατότητες για τα γραφήματα όπως την μετατροπή προς διάφορες μορφές, την δυνατότητα κατασκευής τυχαίων γραφημάτων, την εύρεση υπογράφων, την εύρεση γειτνίασης, την υλοποίηση αλγορίθμων, μέχρι και το σχεδιασμό γραφημάτων σε δύο ή και τρεις διαστάσεις, επιπρόσθετα η βιβλιοθήκη είναι κατάλληλη για μεγάλα γραφήματα του πραγματικού κόσμου.

Πακέτο Matplotlib:

Η Matplotlib είναι μια βιβλιοθήκη της Python όπου συνήθως χρησιμοποιείται σε συνεργασία με την βιβλιοθήκη NumPy της Python, επειδή τις περισσότερες φόρες χρειάζεται για το κομμάτι των μαθηματικών που χρησιμοποιούνται στα γραφήματα. Παράλληλα η βιβλιοθήκη μας παρέχει την δυνατότητα για την δημιουργία δισδιάστατων γραφικών, που παράγει υψηλής ποιότητα γραφικά. Τα γραφικά αυτά έχουν διάφορους τύπους, δημιουργούνται εύκολα και μπορούν να αποθηκευτούν σε αρχεία διαφόρων τύπων. Το πακέτο χρησιμοποιήθηκε στην υλοποίηση για να αναπαριστάνει κάθε φορά ένα διαφορετικό γράφημα.

Πακέτο time:

Η time είναι μια βιβλιοθήκη της Python που δίνει την δυνατότητα να παρέχει πολλούς τρόπους αναπαράστασης του χρόνου σε κώδικα όπως αντικείμενα, αριθμούς, και συμβολοσειρές. Παρέχει επίσης και άλλες λειτουργίες όπως την αναπαράσταση του χρόνου, την αναμονή κατά την εκτέλεση του κώδικα, άλλα και την μέτρηση της αποτελεσματικότητας του κώδικα.

Πακέτο platform:

Η platform είναι μια βιβλιοθήκη της Python όπου χρησιμοποιείται για να μπορεί να μας παρέχει όσο το δυνατόν περισσότερες πληροφορίες σχετικά με το υπολογιστικό σύστημα στο οποίο εκτελείται το πρόγραμμα εκείνη την χρονική στιγμή. Τώρα ως πληροφορίες ορίζουμε τις πληροφορίες της συσκευής όπως το λειτουργικό σύστημα, την έκδοση του λειτουργικού συστήματος, την έκδοση της Python, τον επεξεργαστή που τρέχει στο υπολογιστικό σύστημα. Συγκεκριμένη βιβλιοθήκη είναι σημαντική διότι μας δίνει την δυνατότητα να μπορούμε να ελέγξουμε εάν το σύστημα μας πληροί τις απαιτήσεις του προγράμματος μας.

Πακέτο copy:

Η copy αποτελεί μια βιβλιοθήκη της Python όπου χρησιμοποιείται για να μπορεί ο χρήστης να δημιουργήσει πραγματικά αντίγραφα αυτή η ενέργεια γίνεται με δύο τρόπους το ρηχό αντίγραφο *(Shallow Copy),* και το βαθύ αντίγραφο *(Deep Copy).* Με την δημιουργία ενός ρηχού αντιγράφου δημιουργείται ένα νέο αντικείμενο όπου αποθηκεύει την αναφορά των αρχικών στοιχείων. Ενώ με την δημιουργία ενός βαθύ αντιγράφου δημιουργεί ένα νέο αντικείμενο και προσθέτει αναδρομικά τα αντίγραφα των αντικειμένων που υπήρχαν στα πρωτότυπα στοιχεία.

Πακέτο sys:

Η sys είναι μια βιβλιοθήκη της Python όπου παρέχει στον χρήστη διάφορες συναρτήσεις και μεταβλητές που χρησιμοποιούνται για τον χειρισμό διαφορετικών τμημάτων του χρόνου εκτέλεσης του περιβάλλοντος (*runtime environment).* Επίσης επιτρέπει την λειτουργία στον διερμηνέα *(interpreter)* να παρέχει πρόσβαση στις μεταβλητές και τις συναρτήσεις που αλληλοεπιδρούν με τον διερμηνέα.

Πακέτο numpy:

Η numpy είναι μια βιβλιοθήκη της Python όπου αποτελεί βασική υποδομή για επιστημονικές εφαρμογές καθώς υποστηρίζει εύκολα και πλήρως πολυδιάστατους πίνακες και συναρτήσεις γραμμικής άλγεβρας. Η βιβλιοθήκη διευκολύνει τον χρήστη του στο να κάνει εύκολους υπολογισμούς με πίνακες, να διαβάσει και να γράφει σύνολα δεδομένων και να συνεργάζεται με άλλες γλώσσες προγραμματισμού .

Πακέτο os:

Η os είναι μια βιβλιοθήκη της Python που παρέχει στον χρήστη λειτουργίες για αλληλεπίδραση με το λειτουργικό σύστημα. Ταυτόχρονα η βιβλιοθήκη περιλαμβάνει πολλές λειτουργίες που οδηγούν σε μια αλληλεπίδραση με το σύστημα αρχείων *(files system)*.

4.2 Ανάλυση Υλοποίησης

Στην υποενότητα αυτήν παρουσιάζονται οι βασικοί αλγόριθμοι που αναπτύχθηκαν με τα αντίστοιχα προγραμματιστικά εργαλεία. Για αρχή θα πρέπει να επισημανθεί ότι δημιουργήθηκαν 4 αρχεία για την ανάλυση της υλοποίησης. Τα αρχεία μας επιγραμματικά είναι η υλοποίηση του αλγορίθμου Stoer Wagner, η υλοποίηση του αλγορίθμου Ford Fulkerson, η υλοποίηση του αλγορίθμου Stoer Wagner μέσα από το πρίσμα της βιβλιοθήκης NetworkX. Ο σκοπός της δημιουργίας των δύο τελευταίων αρχείων είναι για να μπορούμε να επιβεβαιώσουμε την σωστή λειτουργία της υλοποίησης του αλγορίθμου Stoer Wagner.

Αλγόριθμος Ford – Fulkerson :

```
#------------------Class BFS-------------------------
'''''
class BFS:

  def bfs (self, graph, s, t, parent):
    visited = [False] * len(graph)
    queue = []
    queue. append(s)
    visited[s] = True
    while queue:
      u = queue.pop (0)
      for ind, val in enumerate(graph[u]):
        if val > 0 and visited[ind] == False:
          queue. append(ind)
          visited[ind] = True
          parent[ind] = u
    if visited[t]:
      return True
    return False

#------------------corteSt Function -------------------------
```

```
def MaxFLowLST(results):
    global id
    value = float('inf')
    for i, maxFlow in enumerate(results):
        if maxFlow [2] < value:
            value = maxFlow [2]
            id = i
    return results[id]
#------------------ Set Function -------------------------

def Set (Flow, graph):
    VertexLst = list ()
    for i in range(graph.row):
        value = 0
        for j in range (graph. column):
            value = value + graph. graph_[i][j]
        if value == Flow [2]:
            VertexLst.append(i)
    return VertexLst.
#------------------Ford Fulkerson Function -------------------------
def FordFulkerson (graph, s, terminal, search):
    parent = [-1] * graph.row
    maxFlow = 0
    list1=list ()
    list2=list ()
    while search.bfs (graph.graph, s, terminal, parent):
        t = terminal
        path = float('inf')
        while t! = s:
            t_ = parent[t]
            path = min (path, graph.graph[t_][t])
            t = parent[t]
        maxFlow += path
        v = terminal
        while (v! = s):
            u = parent[v]
            graph. graph[u][v] -= path
            graph. graph[v][u] += path
            v = parent[v]
    for i in range(graph.row):
        for j in range (graph. column):
            if graph. graph_[i][j] > 0 and graph.graph[i][j] == 0:
                list1.append(i)
                list2.append(j)
    return list1, list2, maxFlow.
```

Εξήγηση Αλγορίθμου :

Η συνάρτηση Ford Fulkerson δέχεται ως είσοδο το γράφημα που την έχουμε ορίζει ως μια κλάση *(class)* που δεν φαίνεται στο παραπάνω κώδικα. Στην κλάση αρχικοποιούμε τις κορυφές, τις ακμές, το πως θα διαβάσει γραμμή το σύνολο δεδομένων, μέχρι ακόμα και την κορυφή πηγής *(source)* και την τερματική κορυφή *(sink)*. Όλα αυτά που αναφέραμε τα αρχικοποιούμε μέσω της μεθόδου __init__ που είναι μια ειδική μέθοδος που διαθέτει η Python για την κατασκευή *(construction)* και την αρχικοποίηση *(initialization)* νεών αντικειμένων, πέρα όμως από το γράφημα η συνάρτηση έχει ως όρισμα τον αρχικό κόμβο *(s)*, αλλά και τον τερματικό *(t)* όπου κάθε φορά είναι διαφορετικός. Πρώτου όμως εμβαθύνουμε στην συνάρτηση Ford Fulkerson θα πρέπει να ορίσουμε την κλάση BFS και τις βοηθητικές συναρτήσεις *(MaxFlowLst)* και *(Set)*. Αρχικά η BFS έχει μια συνάρτηση bfs όπου κάνει στο γράφημα που έχει σαν όρισμα αναζήτηση κατά πλάτος δηλαδή έχοντας σαν αφετηρία τον αρχικό κόμβο *(s)* ξεκινά και διερευνά πρώτα τους γειτονικούς κόμβους, και μετέπειτα μεταβαίνει στους γείτονες του επόμενου επιπέδου, σκοπός μια τέτοιας συνάρτησης είναι μέσω του BFS να βρει διαδρομές που θα υπολογίζουν την μέγιστη ροή. Όσο αναφορά την βοηθητική συνάρτηση MaxFlowLst η συγκεκριμένη συνάρτηση έχει σαν όρισμα μια λίστα *(results)* όπου αποθηκεύει κάθε φορά στην συγκεκριμένη λίστα την μέγιστη ροή που προκύπτει από την κάθε διαδρομή με σταθερό αρχικό κόμβο και διαφορετικό τερματικό κόμβο. Η συνάρτηση Set περιέχει σαν ορίσματα ένα *(Flow)* και ένα γράφημα *(graph)* που είναι ένα αντικείμενο. Η συνάρτηση επιστρέφει μια λίστα *(VertexLst)* με τις κορυφές που έχει περάσει μια διαδρομή και κάθε φορά γίνεται έλεγχος. Για το τέλος όπως αναφέραμε και πάνω υπάρχει η συνάρτηση Ford Fulkerson που επιστέφει την μέγιστη ροή από το s στο t σε ένα δοθέν γράφημα. Ορίζουμε ένα πίνακα *(Parent)* που συμπληρώνεται από την συνάρτηση BFS για αποθήκευση διαδρομής . Στην επανάληψη του While αυξάνουμε την ροή σε περίπτωση που υπάρχει διαδρομή από το s στο t, και βρίσκει την ελάχιστη υπολειπόμενη χωρητικότητα των ακμών, αφού η διαδρομή συμπληρωθεί από το BFS τότε είμαστε σε θέση να βρούμε την μεγίστη ροή. Στο τέλος της συνάρτηση προσθέτουμε την ροή της διαδρομής στην συνολική ροή και ενημερώνουν τις υπολειπόμενες χωρητικότητες των ακμών και των αντίστροφων ακμών κατά μήκος του μονοπατιού.

Αλγόριθμος Stoer – Wagner :

```
#------------------MinimumCutPhase Function-------------------------
def MinimumCutPhase (G, w, a):
   A = list ()
   A = [a]
   B = copy. deepcopy (G. Vertex)
```

```
    B. remove(a)
    dict1 = dict ()
    for v in B:
        dict1[v] = G.G[v, a]
    while len(A)! = len (G. Vertex):
        s = 0
        d = 0
        for i, j in dict1.items():
            if j>d:
                d = j
                s = i
            else:
                continue
        del dict1[s]
        for i, j in dict1.items():
            j += G.G[i, s]
            dict1[i] = j
        A. append(s)
        B. remove(s)
    h=G.G[A [-1],]
    cut = sum(h)

    for i, j in enumerate(G.G[A[-1],]):
        if j!= 0 and i != A[-2]:
            G.G[A[-2], i] =G.G[A[-2],i] + j
            G.G[i, A[-2]] =G.G[i,A[-2]] + j
        G.G[A[-1], i] = 0
        G.G[i, A[-1]] = 0
    G. Vertex.remove(A [-1])
    return cut, A


#------------------MinimumCut Function---------------------------------
def MinimumCut (G, w, a):
    global MinCut
    while (len (G. Vertex)) > 1:
        z = MinimumCutPhase (G, w, a)
        if z [0] < MinCut:
            MinCut = z [0]
        else:
            continue
    return MinCut.
```

Εξήγηση Αλγορίθμου : Για τον αλγόριθμο έχουμε 2 συναρτήσεις την MinimumCutPhase και την MinimumCut. Όπως και στην υλοποίηση του Ford Fulkerson έτσι και στην

υλοποίηση του Stoer Wagner ορίζουμε μια κλάση *(class)* όπου μέσω της μεθόδου __init__ κατασκευάζουμε και αρχικοποιούμε νέα αντικείμενα όπως το την δομή του γραφήματος, τις κορυφές, τις ακμές. Η συνάρτηση MinimumCutPhase έχει ως ορίσματα το γράφημα G, το w που περιγράφει το άθροισμα των βαρών όλων των ακμών και το a που είναι κάθε φορά μια διαφορετική κορυφή. Ορίζουμε μια κενή λίστα A και μια λίστα B (δημιουργεί ένα βαθύ αντίγραφο για τις κορυφές. Έχουμε μια αρχική κορυφή a δημιουργείται μια τακτοποίηση γειτνίασης A που περιέχει την κορυφή a. Κατά την διάρκεια της συνάρτησης στην λίστα A προθέτεται η κορυφή που έχει μεγαλύτερη συνδεσιμότητα με την λίστα A και μέσω της μεθόδου append αποθηκεύεται η τομή της κάθε φάσης στην λίστα μετά από διαδοχικούς ελέγχους. Στο τέλος της συνάρτησης με μια δομή επανάληψης και τους αντίστοιχους ελέγχους συρρικνώνουμε το γράφημα G με τις δύο τελευταίες κορυφές της λίστας A. Η συνάρτηση MinimumCut έχει τα ίδια ορίσματα με την MinimumCutPhase, η συνάρτηση κάνει έναν έλεγχο εάν το σύνολο των κορυφών του γραφήματος είναι μεγαλύτερο της μονάδας εφόσον είναι καλούμε την συνάρτηση MinimumCutPhase και εκτελούμε έναν έλεγχο ότι αν το cut είναι μικρότερο από το τρέχον τότε θέτουμε το cut ως την ελάχιστη τομή του γραφήματος.

Κεφάλαιο 5. Πειράματα

Στην ενότητα αυτή συνοψίζουμε και παρουσιάζουμε τα πειράματα της διατριβής μας

5.1 Περιγραφή Πειραμάτων

Αφού υλοποιήσαμε τους αλγορίθμους Stoer Wagner, τον Ford Fulkerson, αλλά και τον Stoer Wagner μέσα από το πρίσμα της βιβλιοθήκης network, είμαστε σε θέση να υλοποιήσουμε τα αντίστοιχα πειράματα. Αρχικά σαν στάδιο προ επεξεργασίας θα πρέπει να τρέξουμε τον αλγόριθμο που θέλουμε στην κονσόλα μας και να περάσουμε σαν είσοδο *(input)* ένα αρχείο text. Για να μας βγάλει το προβλεπόμενο αποτέλεσμα θα πρέπει η δομή του text να είναι η εξής: vertex1 vertex2 weight όπου έχουμε δύο κορυφές που ενώνονται υπό το βάρος *(weight)* μιας ακμής . Για τον αλγόριθμό Ford Fulkerson και μόνο η δομή του text είναι κάπως παραλλαγμένη δηλαδή είναι ακριβώς ίδια με την παραπάνω με την μόνη διαφορά ότι στην πρώτη γραμμή του κάθε text θα πρέπει να αναφέρεται ο ακριβής αριθμός των κορυφών και των ακμών. Ενδεικτικά για τα πειράματα έχουμε 15 text αρχεία ταξινομημένα που τα εκτελούμε για κάθε αλγόριθμο και μας βγάζει σαν αποτέλεσμα την ελάχιστή τομή υπό μορφή ακεραίου αριθμού και ταυτόχρονα το χρόνο που έκανε να εκτελεστεί η κάθε συνάρτηση.

5.2 Πειράματα

Σε αυτό το πείραμα έχουμε 15 σύνολα δεδομένων για τις 3 υλοποιήσεις και τις συγκρίνουμε με βάση το χρόνο, Παράλληλα υπάρχουν και 3 γραφικές παραστάσεις για τον κάθε αλγόριθμο σε μορφή Χρόνος/Κορυφή όπου μπορούμε να αντλήσουμε συμπεράσματα για την συμπεριφορά του κάθε αλγορίθμου.

Ford Fulkerson:

#	Vertex	Edges	Time(ns)	Time(s)	Result
Test1.txt	18	55	4000000	0.004	6882
Test2.txt	21	25	4000000	0.004	1526
Test3.txt	41	52	12000000	0.012	648
Test4.txt	51	98	42000000	0.042	377
Test5.txt	101	198	1674000000	1.674	8023

Test6.txt	151	196	745000000	0.745	1153
Test7.txt	165	305	3310000000	3.31	95
Test8.txt	401	528	25060000000	25.06	904
Test9.txt	451	594	31420000000	31.42	336
Test10.txt	501	5961	33270000000	33.27	380
Test11.txt	502	668	39830000000	39.83	584
Test12.txt	504	676	40240000000	40.24	805
Test13.txt	971	5656	9.843E+11	984.3	6872
Test14.txt	971	7757	1.492E+12	1492	9369
Test15.txt	1001	5001	1.38E+12	1600	36

Εικόνα 5.1

Stoer Wagner NetworkX:

#	Vertex	Edges	Time(ns)	Time(s)	Result
Test1.txt	18	55	297000000	0.297	6882
Test2.txt	21	25	265000000	0.265	1526
Test3.txt	41	52	296000000	0.296	648
Test4.txt	51	98	297000000	0.297	377
Test5.txt	101	198	359000000	0.344	8023
Test6.txt	151	196	344000000	0.359	1153
Test7.txt	165	305	391000000	0.391	95
Test8.txt	401	528	707000000	0.707	904
Test9.txt	451	594	847000000	0.847	336
Test10.txt	501	5961	3333000000	3.333	380
Test11.txt	502	668	998000000	0.998	584
Test12.txt	504	676	1022000000	1.022	805
Test13.txt	971	5656	7854000000	7.854	6872

Test14.txt	971	7757	9951000000	9.951	9369
Test15.txt	1001	5001	10840000000	10.84	36

Εικόνα 5.2

Stoer Wagner:

#	Vertex	Edges	Time(ns)	Time(s)	Result
Test1.txt	18	55	1000000	0.001	6882
Test2.txt	21	25	1000000	0.001	1526
Test3.txt	41	52	12000000	0.012	648
Test4.txt	51	98	15000000	0.015	377
Test5.txt	101	198	78000000	0.078	8023
Test6.txt	151	196	218000000	0.218	1153
Test7.txt	165	305	344000000	0.344	95
Test8.txt	401	528	4089000000	4.089	904
Test9.txt	451	594	5078000000	5.078	336
Test10.txt	501	5961	6858000000	6.858	380
Test11.txt	502	668	7520000000	7.52	584
Test12.txt	504	676	7620000000	7.62	805
Test13.txt	971	5656	47910000000	47.91	6872
Test14.txt	971	7757	48250000000	48.25	9369
Test15.txt	1001	5001	58980000000	58.98	36

Εικόνα 5.3

Εκτελώντας τους δυο αλγορίθμους που έχουμε εφαρμόσει στα γραφήματα του συνόλου δεδομένων είμαστε σε θέση να βγάλουμε κάποια συμπεράσματα. Τα αποτέλεσμα της ελάχιστης τομής για κάθε σύνολο δεδομένων αποτυπώνονται μέσα από τις γραφικές παραστάσεις για την κάθε υλοποίηση του αλγόριθμου. Σε αυτό το σημείο είμαστε σε θέση να απαντήσουμε σε κάποιες ερωτήσεις που θέτουμε σαν χρήστες.

Ερώτηση 1: Πως συμπεριφέρονται οι αλγόριθμοι σε σχέση με τις διάφορες περιπτώσεις; Υπάρχει ένας αλγόριθμος που είναι πάντα καλύτερος από τον άλλο; Ποιος αλγόριθμος είναι πάντα πιο αποτελεσματικός ;

Όταν έχουμε σύνολα δεδομένων με μικρές κορυφές όπως τα (test1, test2, test3, test4, test5, test6, test7) και οι 3 υλοποιήσεις έχουν παρόμοιους χρόνους όντας η υλοποίηση Ford Fulkerson λίγο πιο αργή χρονικά στα όρια όμως του μαθηματικού σφάλματος. Εξάγοντας κάποια δεδομένα από τα διαγράμματα συμπεραίνει κάνει ότι από το σύνολο δεδομένων test8, ο χρόνος του Ford Fulkerson αυξάνεται απότοπα (μπορεί να το παρατηρήσεις κάποιος και στο διάγραμμα). Άρα μπορούμε να συμπεράνουμε ότι όσο αυξάνονται οι κορυφές τόσο ο αλγόριθμος Ford Fulkerson γίνεται χρονικά πιο αργός και με μεγαλύτερη πολυπλοκότητα. Απαντώντας στο βασικό ερώτημα σχετικά με την πολυπλοκότητα και την απλότητα ο Ford Fulkerson είναι πιο περίπλοκος, ενώ ο Stoer Wagner που χαρακτηρίζεται από τις δύο υλοποιήσεις είναι πιο απλός, κάτι που

χαρακτηρίζεται από τους ίδιους τους συγγράφεις της διατριβής με την ονομασία απλός αλγόριθμος ελάχιστης τομής.

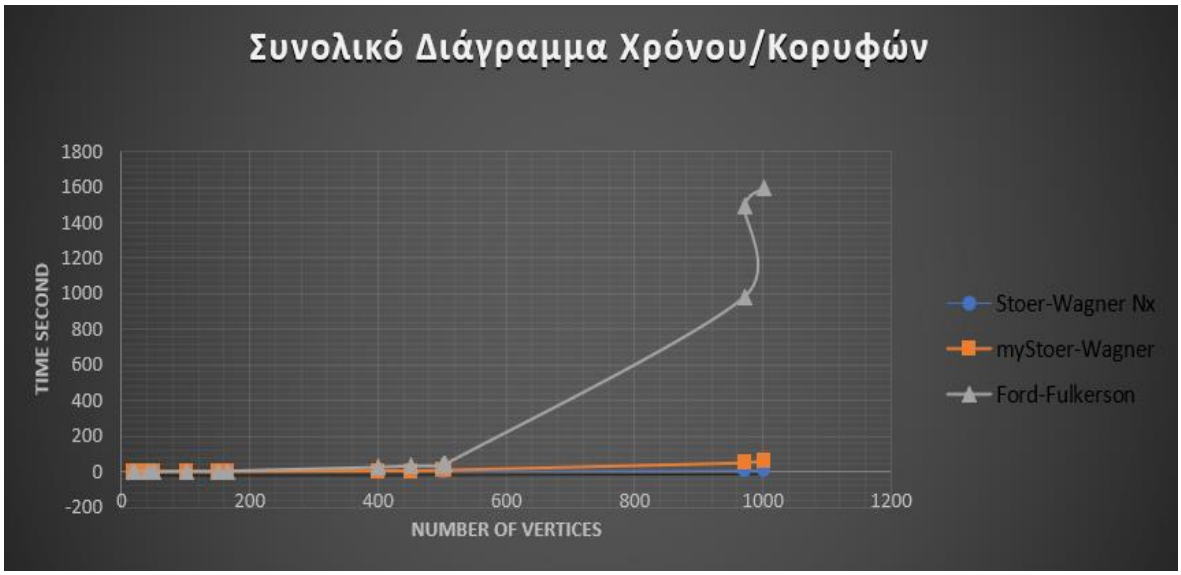
Ερώτηση 2: Με δεδομένο ότι ο αλγόριθμός Stoer Wagner είναι καλύτερος και πιο αποδοτικός ποια από τις 2 υλοποιήσεις είναι καλύτερη η networkX ή myStoer Wagner ;

Στηριζόμενοι στα δεδομένα των γραφικών παραστάσεων μπορούμε να συμπεράνουμε ότι η υλοποίηση της networkX είναι καλύτερη από την myStoer Wagner ο λόγος που συμβαίνει αυτό δεν είναι ο προφανής, δηλαδή δεν είναι ο αλγόριθμος, διότι και οι δυο υλοποιήσεις βασίζονται στον ίδιο αλγόριθμο. Η ειδοποιός διαφορά στις δύο υλοποιήσεις ανάγεται στο προγραμματιστικό κομμάτι δηλαδή στην υλοποίηση του networkX χρησιμοποιούνται καλύτερες προγραμματιστικές τεχνικές όπως την αποφυγή περιττών επαναλήψεων, την χρησιμοποίηση της αναδρομής, αλλά και καλύτερή χρησιμοποίηση δομών δεδομένων όπως για παράδειγμα μια λίστα. Αντίθετα στην υλοποίηση του myStoer Wagner κάνουν χρήση δομών δεδομένων με περιορισμένο χώρο μνήμης και με δύστροπη μεταχείριση όπως λεξικά και σύνολα. Συνοψίζοντας θα μπορούσαμε να αναφέρουμε εκλαϊκευμένα ότι στην υλοποίηση της networkX έχει γραφτεί ένας πολύ καλύτερος και δομημένος κώδικας σε σχέση με την myStoer Wagner.

Στο παρακάτω πίνακα φαίνονται οι χρόνοι και των τριών υλοποιήσεων για να φαίνεται πιο ξεκάθαρα τα συμπεράσματα που αναφέραμε παραπάνω.

	Ford-Fulkerson	Stoer-Wagner NetworkX	Stoer-Wagner
#	Time(s)	Time(s)	Time(s)
Test1.txt	0.004	0.297	0.001
Test2.txt	0.004	0.265	0.001
Test3.txt	0.012	0.296	0.012
Test4.txt	0.042	0.297	0.015
Test5.txt	1.674	0.344	0.078
Test6.txt	0.745	0.359	0.218
Test7.txt	3.31	0.391	0.344
Test8.txt	25.06	0.707	4.089
Test9.txt	31.42	0.847	5.078
Test10.txt	33.27	3.333	6.858
Test11.txt	39.83	0.998	7.52
Test12.txt	40.24	1.022	7.62
Test13.txt	984.3	7.854	47.91
Test14.txt	1492	9.951	48.25
Test15.txt	1600	10.84	58.98

Στο παρακάτω διάγραμμα αναπαριστούμε συνολικά και τις 3 υλοποιήσεις Χρόνου/Αριθμός κορυφών και φαίνεται ποιος είναι πιο αποτελεσματικός με βάση το χρόνο.



Εικόνα 5.4

Για το τέλος το τελευταίο πείραμα που ολοκληρώθηκε ήταν το εξής: χρησιμοποιώντας την βιβλιοθήκη networkX δημιουργήσαμε 5 τυχαία σύνολα δεδομένων για μη κατευθυνόμενα γραφήματα και τα χρησιμοποιήσαμε και για τις 3 υλοποιήσεις. Στους παρακάτω πίνακες αναφέροντα τα σύνολα δεδομένων και τα αποτελέσματά τους για κάθε υλοποίηση.

#	Vertex	Edges	Result
Test1.txt	50	1225	1005
Test2.txt	100	4950	4311
Test3.txt	200	19900	17129
Test4.txt	300	44850	40904
Test5.txt	400	73257	79800

	Ford-Fulkerson	Stoer-Wagner NetworkX	Stoer-Wagner
#	Time(s)	Time(s)	Time(s)
Test1.txt	0.471	0.270	0.043
Test2.txt	7.286	0.481	0.041
Test3.txt	118.3	1.424	9.947
Test4.txt	620.1	4.606	55.12
Test5.txt	2530	161.3	165.9

Κεφάλαιο 6. Επίλογος

Στην ενότητα αυτή συνοψίζουμε τη συνεισφορά και τα αποτελέσματα της διατριβής μας και παραθέτουμε σκέψεις για μελλοντικές επεκτάσεις της.

6.1 Σύνοψη και συμπεράσματα

Στην παρούσα διπλωματική διατριβή μελετήσαμε ένα διαχρονικό πρόβλημα της θεωρίας γραφημάτων το πρόβλημα της ελάχιστης τομής ενός γραφήματος. Παρατηρήσαμε ότι το συγκεκριμένο πρόβλημα ανάγεται σε πολλές εφαρμογές. Παράλληλα μελετήσαμε έναν από τους πιο βασικούς αλγόριθμους ελάχιστης τομής με τεχνικές ροής, τον αλγόριθμο Ford Fulkerson και το εφαρμόσαμε σε ένα τυχαίο γράφημα. Επίσης αναπτύξαμε τον πιο βασικό αλγόριθμο της διατριβής μας τον αλγόριθμο Stoer Wanger χρησιμοποιώντας την τεχνική διαδοχικής συρρίκνωσης ακμών και το εφαρμόσαμε και αυτόν σε ένα τυχαίο γράφημα. Εφαρμόσαμε και στους δύο αλγόριθμους κάποια σύνολα δεδομένων και μελετήσαμε την συμπεριφορά τους. Παρατηρήσαμε ότι ο αλγόριθμος Stoer Wagner που αναπτύξαμε έχει μικρότερη χρονική πολυπλοκότητα από τον αλγόριθμο Ford Fulkerson άρα μπορέσαμε να βγάλουμε το συμπέρασμα ότι είναι αρκετά πιο γρήγορος, πιο αποδοτικός, και σχεδόν πάντα καλύτερος από τον αλγόριθμο Ford Fulkerson. Βγάζοντας λοιπόν ένα γενικό συμπέρασμα από την διατριβή μας στην σύγχρονη εποχή όπου το πρόβλημα της ελάχιστης τομής υπάγεται σε πολλές εφαρμογές ο αλγόριθμος Stoer Wagner μπορεί να είναι ο κύριος εκφραστής της λύσης του συγκεκριμένου προβλήματος.

6.2 Μελλοντικές επεκτάσεις

Για το τέλος θα ήταν σημαντική παράλειψη να μην αναφέρουμε τις μελλοντικές επεκτάσεις του προβλήματος της ελαχίστης τομής μέσα από την σκοπιά του αλγορίθμου Stoer Wagner. Καταρχήν ένα σημαντικό ερώτημα που τίθεται είναι κατά πόσο θα μπορούσαμε να βελτιώσουμε την χρονική πολυπλοκότητα του αλγορίθμου αποδοτικά. Σε περιπτώσεις πολύ μεγάλων γραφημάτων ο αλγόριθμος μπορεί να είναι αργός, μπορεί όμως να αναπτυχθούν παράλληλοι και κατανεμημένοι αλγόριθμοι που θα μπορούσαν να αξιοποιήσουν τον αλγόριθμο Stoer-Wagner για να υπολογίζει την ελάχιστη τομή σε πολύ μεγάλα γραφήματα. Ο αλγόριθμος Stoer-Wagner έχει σχεδιαστεί

για στατικά γραφήματα αλλά όπως και σε πολλές εφαρμογές του πραγματικού κόσμου, τα γραφήματα μπορούν να αλλάξουν δυναμικά με την πάροδο του χρόνου. Θα μπορούσε λοιπόν να γινεί μια έρευνα βασισμένο στον αλγόριθμο μας για ανάπτυξη νέων αλγορίθμων που θα μπορούν να χειριστούν δυναμικά γραφήματα και να ενημερώνουν έγκαιρα για την ελάχιστη τομή όταν το γράφημα αλλάζει μορφή. Συμπερασματικά ο αλγόριθμος Stoer Wagner αν και σε μεγάλό βαθμό είναι γρήγορος επιδέχεται περιθώρια βελτίωσης και απόδοσης.

6.3 Βιβλιογραφικές Αναφορές - References

[SW 97] M. Stoer and F. Wagner. A simple min-cut algorithm. Journal of the ACM, 44(4):585–591, 1997.

[FD 56] L. R. Ford, Jr, and D.R. Fulkerson. ‘’Maximal Flow Through a Network’’. Canadian Journal of Math... 8: 399-404, 1956

[NI 92] H. Nagamochi and T. Ibaraki. ‘’Computing edge-connectivity in multigraphs and capacitated graphs’’, SIAM Journal on Discrete Mathematics, 5: 54-66, 1992.

[KS 93] Karger, D., and Stein, C. 1993. An O(n^2) algorithm for minimum cuts. In Proceedings of the 25th ACM symposium on the Theory of Computing (San Diego, Calif, May 16-18). ACM, New York, pp. 757-765.

[NS 89] Nishizeki, T., and Poljak, S. 1989. Highily connected factors with a small number of edges. Preprint.

[KB 06] Kleinberg, J., and Tardos, E. 2006. Algorithm Design. University Cornell

[R 19] Kenneth Rosen. 2019 Discrete Mathematics and its applications. McGraw-Hill, New York.

[CLRS 11] Thomas H. Cormen, Charlese Leiserson, Ronald Rivest, Clifford Stein, Introduction to algorithms. MIT Press

[Q 95] Queyranne, M. 1995. A combinatorial algorithm for minimizing symmetric submodular functions. In Proceedings of the 6th ACM-SLAM Symposium on Discrete Mathematics ACM, New York, pp. 98-101.

WIKIPEDIA: The Free Encyclopedia. In www.wikipedia.org (2001).

6.4 Μεταφράσεις Ξένων Όρων

Μετάφραση	Αγγλικός όρος
Ακμές	edges
Αλγόριθμος	algorithm
Βάρος	weight
Γράφημα	graph
Ελάχιστη τομή	minimum cut
Κορυφή	vertex
Πακέτο	package
Πίνακας	array
Πίνακας γειτνίασης	adjacency matrix
Ροή	flow
Χωρητικότητα	capacity
Συρρικνωμένο γράφημα	contracted graph