ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Εύρεση Επικαλυπτόμενων Κοινοτήτων με τη χρήση της βιβλιοθήκης JGraphT

Πτυχιακή Εργασία

Φώτης Καραγιάννης

Αθήνα, 2021-2022

HAROKOPIO UNIVERSITY

SCHOOL OF DIGITAL TECHNOLOGY

INFORMATICS AND TELEMATICS

Graph Algorithms in JGraphT

Bachelor Thesis

Fotis Karagiannis

Athens, 2021-2022

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

ΣΧΟΛΗ ΨΗΦΙΑΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΜΑΤΙΚΗΣ

Τριμελής Εξεταστική Επιτροπή

Επιβλέπων Πτυχιακής Εργασίας

Δημήτριος Μιχαήλ

Αναπληρωτής Καθηγητής, τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Μέλος Τριμελούς Επιτροπής

Μαλβίνα Βαμβακάρη

Καθηγήτρια, τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Μέλος Τριμελούς Επιτροπής

Ηρακλής Βαρλάμης

Αναπληρωτής Καθηγητής, τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο

Ο Φώτης Καραγιάννης

Δηλώνω υπεύθυνα ότι:

1. Είμαι ο κάτοχος των πνευματικών δικαιωμάτων της πρωτότυπης αυτής εργασίας και από όσο γνωρίζω η εργασία μου δε συκοφαντεί πρόσωπα, ούτε προσβάλει τα πνευματικά δικαιώματα τρίτων.

2. Αποδέχομαι ότι η ΒΚΠ μπορεί, χωρίς να αλλάξει το περιεχόμενο της εργασίας μου, να τη διαθέσει σε ηλεκτρονική μορφή μέσα από τη ψηφιακή Βιβλιοθήκη της, να την αντιγράψει σε οποιοδήποτε μέσο ή/και σε οποιοδήποτε μορφότυπο καθώς και να κρατά περισσότερα από ένα αντίγραφα για λόγους συντήρησης και ασφάλειας.

3. Όπου υφίστανται δικαιώματα άλλων δημιουργών έχουν διασφαλιστεί όλες οι αναγκαίες άδειες χρήσης ενώ το αντίστοιχο υλικό είναι ευδιάκριτο στην υποβληθείσα εργασία

Η συγκεκριμένη πτυχιακή εργασία είναι αφιερωμένη στην οικογένεια μου που με στήριξε στην διάρκεια εκπόνησης της αλλά και σε όλη την διάρκεια φοίτησης μου.

“An algorithm must be seen to be believed.”

Donald Ervin Knuth, The Art of Computer Programming

Περιεχόμενα

Κατάλογος Εικόνων

Οι περισσότερες εικόνες με εξαίρεση τα λιγοστά σχήματα που παράχθηκαν με την χρήση του εργαλείου ανοιχτού λογισμικού Graphviz δημιουργήθηκαν αποκλειστικά για τις ανάγκες της συγκεκριμένης πτυχιακής εργασίας.

Κατάλογος Πινάκων

Περίληψη

Για τους σκοπούς της παρούσας πτυχιακής εργασίας πραγματοποιήθηκε η μελέτη γράφων και αλγορίθμων που στοχεύουν στην εύρεση κοινοτήτων σε αυτούς. Αρχικά, παρουσιάζεται το γενικότερο πλαίσιο που καθιστά τους γράφους μια δομή δεδομένων με πολλές εφαρμογές σε διάφορους επιστημονικούς κλάδους. Στην συνέχεια, παρουσιάζονται και αναλύονται διάφοροι γνωστοί αλγόριθμοι εύρεσης κοινοτήτων που μελετήθηκαν καθώς και το βασικό αντικείμενο μελέτης που είναι η εύρεση επικαλυπτόμενων κοινοτήτων σε γράφους. Στην συνέχεια, αναπτύχθηκε μια υλοποίηση του προηγούμενου αλγορίθμου η οποία αναλύεται σε υψηλό επίπεδο μαζί με τις πρωτοβουλίες και τις επιλογές που προέκυψαν στην διάρκεια σχεδιασμού της. Επιπλέον, αναλύεται και ένας τρόπος εκτέλεσης του αλγορίθμου παράλληλα σε νήματα που προστέθηκε στην τελική υλοποίηση. Τέλος, παρουσιάζονται πειράματα που έγιναν πάνω στο τελικό πρόγραμμα για να δοκιμαστεί η μεταβολή του χρόνου εκτέλεσης στην περίπτωση χρήσης νημάτων, μαζί με πειράματα που έγιναν για να ελεγχθούν τα αποτελέσματα που προκύπτουν γενικότερα από την εκτέλεση του αλγορίθμου.

Λέξεις Κλειδιά: Γράφος, Εύρεση κοινοτήτων, Επικάλυψη Κοινοτήτων, Νήματα

Abstract

For the purpose of the following thesis, a study of graphs and algorithms aimed at finding communities hidden in them was carried out. Firstly, the general reasons making graphs a data structure with many applications in various scientific fields are presented. Following that, various well-known community detection algorithms were studied, presented and analyzed along with the main object of this study, which is detecting overlapping communities inside a graph. Subsequently, an implementation of that algorithm was developed, which is then analyzed at a high level along with the initiatives and choices that emerged during its design. In addition, a way to run the algorithm in parallel fashion on threads that was added in the final implementation is discussed and analyzed. Finally, experiments performed on the final program are presented to test the changes in the execution time depending on the usage of threads, along with experiments performed to test the results obtained from the execution of the algorithm.

Key Words: Graph, Community Detection, Community Overlapping, Threads

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω πρωτίστως τον επιβλέπων καθηγητή Δημήτριο Μιχαήλ για την υποστήριξη, συμβολή αλλά και την συνεχή καθοδήγηση μου κατά την διάρκεια εκπόνησης της συγκεκριμένης εργασίας.

Επιπλέον θα ήθελα να ευχαριστήσω όλους τους καθηγητές και τις καθηγήτριες του τμήματος Πληροφορικής και Τηλεματικής του πανεπιστημίου για τις γνώσεις που μου μετέδωσαν στο διάστημα της φοίτησης μου.

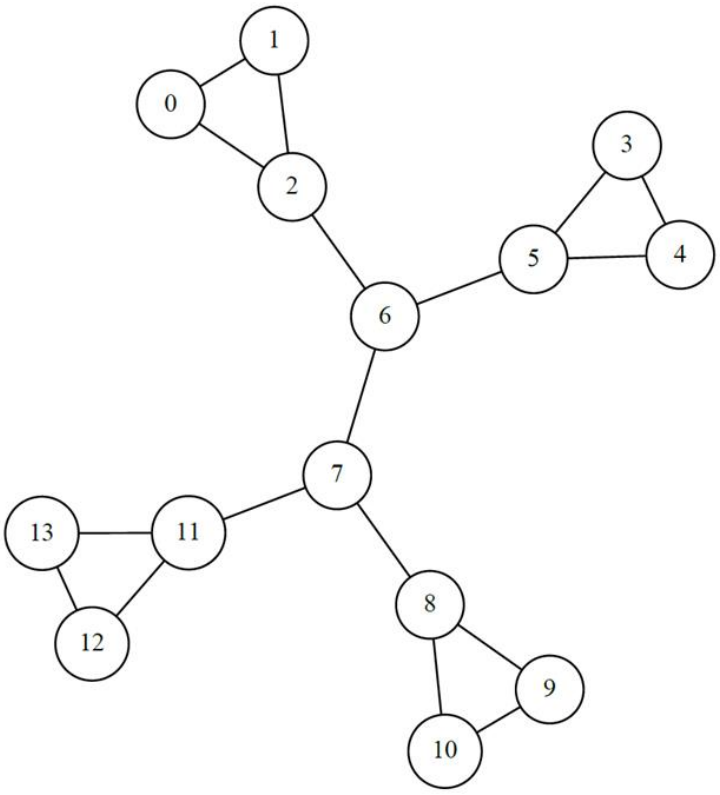
Τέλος, θα ήθελα να ευχαριστήσω τους γονείς και φίλους μου για την συνεχή στήριξη τόσο στην διάρκεια ολοκλήρωσης της συγκεκριμένης πτυχιακής εργασίας όσο και κατά την διάρκεια της φοίτησής μου.

Κεφάλαιο 1ο Εισαγωγή

1.1 Γράφος

Στην σημερινή εποχή, ο κόσμος μας κατακλύζεται από απέραντο όγκο πληροφοριών και δεδομένων. Προκύπτει λοιπόν, περισσότερο από ποτέ, η ανάγκη για ανάπτυξη και εξέλιξη αποδοτικών εργαλείων που βοηθούν στον χειρισμό και στην εξαγωγή χρήσιμης πληροφορίας και συμπερασμάτων από το γενικότερο σύνολο της πληροφορίας. Αδιαμφισβήτητα δομικά στοιχεία που χρησιμοποιούνται για την εξυπηρέτηση αυτού του σκοπού είναι οι δομές δεδομένων και οι αλγόριθμοι που εκτελούνται σε αυτές, από τις οποίες ιδιαίτερο ενδιαφέρον παρουσιάζει ο γράφος.

Γράφος, ή γράφημα, όπως ορίζεται γενικότερα [9], στα διακριτά μαθηματικά είναι μια αφηρημένη αναπαράσταση ενός συνόλου στοιχείων, όπου μερικά από αυτά συνδέονται μεταξύ τους με δεσμούς. Τα διασυνδεδεμένα στοιχεία αναπαρίστανται με μαθηματικές έννοιες οι οποίες ονομάζονται κορυφές ή κόμβοι, ενώ οι δεσμοί μεταξύ των κορυφών ονομάζονται ακμές του γραφήματος. Ένας γράφος μπορεί να αποτυπωθεί διαγραμματικά χρησιμοποιώντας ένα σύνολο κουκκίδων για τις κορυφές και γραμμές ή καμπύλες για τις ακμές.

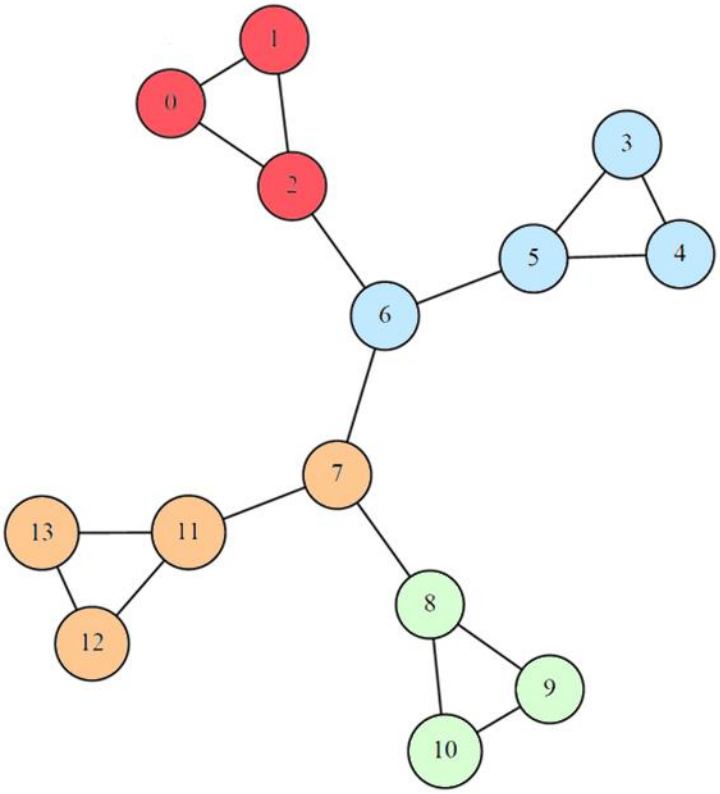


Εικόνα 1: Γραφική Αναπαράσταση Γράφου

Ως εργαλεία, οι γράφοι χρησιμοποιούνται για την αναπαράσταση σχέσεων μεταξύ οντοτήτων και ως αποτέλεσμα βρίσκουν εφαρμογή σε πολλούς επιστημονικούς κλάδους. Συγκεκριμένα, στον κλάδο της πληροφορικής χρησιμοποιούνται για παράδειγμα, στην αναπαράσταση της ροής εκτέλεσης ενός προγράμματος, στην ανάλυση και οργάνωση δεδομένων και δικτύων επικοινωνίας αλλά και στην αναπαράσταση διαδρομών για τις εφαρμογές πλοήγησης αυτοκινήτων. Στις κοινωνικές επιστήμες και ιδιαίτερα στην μελέτη κοινωνικών δικτύων, αναπαριστούν τις κοινότητες ενός συνόλου ανθρώπων αλλά και τις σχέσεις, ψηφιακές ή μη που διατηρούν μεταξύ τους με την χρήση ακμών. Τέλος, στη χημεία και την βιολογία χρησιμοποιούνται για την απεικόνιση και μελέτη των μορίων, γονιδίων αλλά και των πρωτεϊνών.

Μέσω των μερικών μόνο εφαρμογών που αναφέρθηκαν παραπάνω είναι προφανής η σημασία της χρήσης αλλά και της ανάλυσης των γράφων με σκοπό την άντληση πληροφορίας, αλλά και για την πιθανή εξαγωγή συμπερασμάτων για ένα σύνολο ή υποσύνολο των στοιχείων, με βάση των σχέσεων που αναπτύσσονται ή όχι μεταξύ τους. Έτσι προκύπτει η ανάγκη δημιουργίας και χρήσης αυτοματοποιημένων εργαλείων - αλγορίθμων που βοηθούν στην επεξεργασία και μελέτη των γράφων. Μερικοί αλγόριθμοι χρησιμοποιούνται για την εύρεση συντομότερων μονοπατιών μεταξύ δύο κόμβων του γράφου για παράδειγμα οι αλγόριθμοι Dijkstra, Floyd-Warshall, Bellman-Ford κ.ά. Άλλοι χρησιμοποιούνται για την εύρεση ενός ελάχιστα γεννητικού δέντρου ενός γραφήματος, για παράδειγμα οι αλγόριθμοι Kruskal, Prim κ.ά. Ιδιαίτερο ενδιαφέρον έχουν οι αλγόριθμοι εύρεσης κοινοτήτων ενός γράφου, όπου κοινότητα ενός γράφου καλείται ένα γράφημα του οποίου οι κορυφές μπορούν να ομαδοποιηθούν σε ένα υπο-σύνολο κορυφών του αρχικού γραφήματος οι οποίοι είναι εσωτερικά συνδεδεμένες μεταξύ τους.

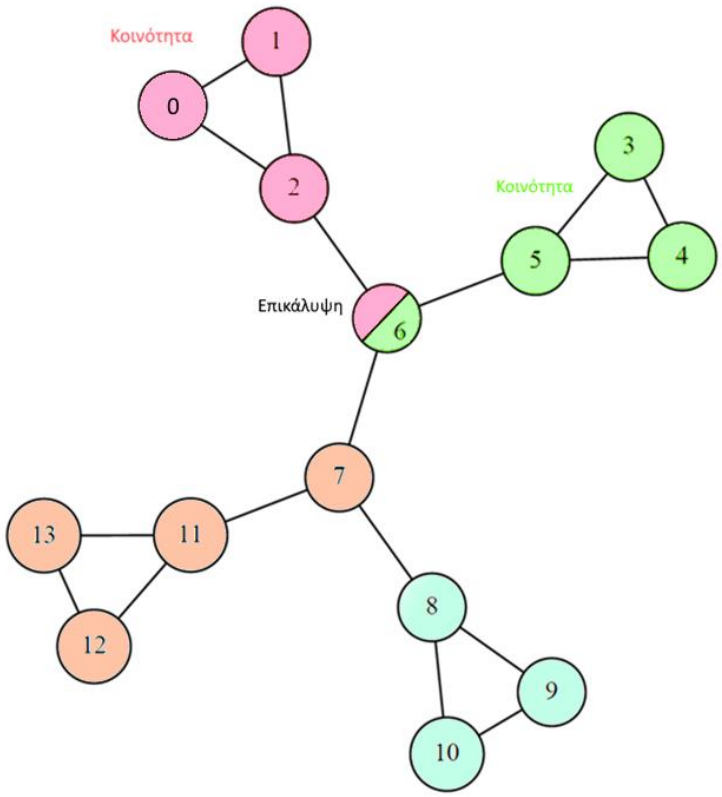
1.2 Κοινότητες Γράφου



Εικόνα 2: Απεικόνιση Κοινοτήτων Γράφου

Οι κοινότητες μπορούν να επικαλύπτουν η μία την άλλη ή και όχι. Οι αλγόριθμοι εύρεσης κοινοτήτων είναι αλγόριθμοι οι οποίοι βοηθούν στην εύρεση όλων των πιθανών κοινοτήτων ενός γράφου. Ως αποτέλεσμα, οι κοινότητες που προκύπτουν βοηθούν στην εξαγωγή συμπερασμάτων για ένα γράφο και κατ' επέκταση για το πραγματικό σύνολο που αναπαριστά. Ένα από τα προβλήματα των περισσότερων παραδοσιακών αλγορίθμων εύρεσης κοινοτήτων, είναι το γεγονός ότι ως προϋπόθεση, θεωρούν ότι το κάθε άτομο-κόμβος του γραφήματος ανήκει το πολύ σε μια κοινότητα την φορά και όχι σε περισσότερες. Σε αντίθεση, στον πραγματικό κόσμο και τα σενάρια που προκύπτουν από αυτόν, κάθε κόμβος μπορεί να ανήκει και σε περισσότερες από μια ομάδες την φορά, κάτι που καλείται επικαλυπτόμενες κοινότητες.

Έτσι δημιουργείται η ανάγκη ανάπτυξης αλγορίθμων που να μπορούν να εντοπίζουν αποδοτικά τις επικαλυπτόμενες κοινότητες ενός γραφήματος από την σκοπιά του κάθε κόμβου ξεχωριστά, ο οποίος μπορεί να ανήκει και σε περισσότερες από μία κοινότητες ταυτόχρονα, με στόχο την ορθότερη μελέτη πραγματικών περιβαλλόντων και σχέσεων από όποιο επιστημονικό πεδίο και αν προέρχονται.



Εικόνα 3: Απεικόνιση Επικάλυψης Κοινοτήτων Γράφου

1.3 Στόχοι Έρευνας

1. Βαθύτερη μελέτη του προβλήματος εύρεσης κοινοτήτων σε γράφους και ειδικότερα κοινοτήτων που επικαλύπτονται μεταξύ τους.

2. Πρακτική σχεδίαση και υλοποίηση του αλγόριθμου εύρεσης επικαλυπτόμενων κοινοτήτων.

3. Μελέτη και ανάπτυξη περιπτώσεων εφαρμογής παράλληλης εκτέλεσης σε νήματα ανεξάρτητων τμημάτων του αλγορίθμου.

4. Ανάπτυξη του κώδικα με τρόπο που θα επιτρέπει την πιθανή ενσωμάτωση στην βιβλιοθήκη JGraphT.

1.4 Περιγραφή Δομής Κεφαλαίων

Η συγκεκριμένη μελέτη, ασχολείται με την υλοποίηση ενός αλγόριθμου εύρεσης επικαλυπτόμενων κοινοτήτων σε γράφους, ο οποίος εντοπίζει φιλικές ομάδες ανάμεσα σε κόμβους που είναι συνδεδεμένοι μεταξύ τους και τις οποίες τελικά οργανώνει σε κοινότητες του γραφήματος που είναι πιθανό να επικαλύπτονται μεταξύ τους. Επιπλέον, γίνεται μια προσπάθεια παραλληλοποίησης των ανεξάρτητων μεταξύ τους τμημάτων του αλγορίθμου με σκοπό την βελτίωση της αποδοτικότητας του. Η γλώσσα προγραμματισμού που χρησιμοποιείται στην υλοποίηση είναι η Java, και παράλληλα χρησιμοποιείται το πακέτο ανοιχτού λογισμικού JGraphT το οποίο περιέχει υλοποιήσεις και εργαλεία σχετιζόμενα με την θεωρία γράφων αλλά και τους αλγόριθμούς που μπορούν να εκτελεστούν σε αυτά για διάφορους σκοπούς. Η παρούσα πτυχιακή οργανώνεται στα παρακάτω κεφάλαια:

Το 1ο Κεφάλαιο αποτελεί μια απλή περιγραφή του γενικότερου ζητήματος των γράφων και των αλγορίθμων που εκτελούνται σε αυτούς, καθώς και της σημαντικότητας τους ως δομή δεδομένων σε μελέτες διαφόρων επιστημονικών κλάδων. Έπειτα, επικεντρώνεται στην σημασία των κοινοτήτων που μπορούν να παρατηρηθούν σε έναν γράφο. Τέλος, παρουσιάζονται οι στόχοι εκπόνησης της πτυχιακής εργασίας και της υλοποίησης του αλγορίθμου επικαλυπτόμενων κοινοτήτων.

Το 2ο Κεφάλαιο εστιάζει στην ανάλυση του υπόβαθρου σχετικά με το θέμα της εύρεσης κοινοτήτων κορυφών σε γράφους και συγκεκριμένα σε υπάρχοντες αλγόριθμους που χρησιμοποιούν ο κάθε ένας την δική του προσέγγιση στην λύση του προβλήματος. Τέλος, γίνεται μια αναλυτική θεωρητική αναφορά στο αντικείμενο μελέτης της συγκεκριμένης πτυχιακής εργασίας, το οποίο είναι η εύρεση επικαλυπτόμενων κοινοτήτων σε γράφους.

Το 3ο Κεφάλαιο επικεντρώνεται στο πρακτικό τμήμα της υλοποίησης του αλγορίθμου επικαλυπτόμενων κοινοτήτων που αναπτύχθηκε για την πτυχιακή εργασία με την βοήθεια του πακέτου JGraphT. Συγκεκριμένα, παρουσιάζονται σε υψηλό επίπεδο ο τρόπος ανάπτυξης του αλγόριθμου σε Java, οι αποφάσεις που πάρθηκαν κατά την διάρκεια υλοποίησης αλλά και τα περαιτέρω βήματα που υλοποιήθηκαν για την βελτίωση του χρόνου εκτέλεσης του τελικού προγράμματος.

Το 4ο κεφάλαιο αφορά τα διάφορα πειράματα που πραγματοποιήθηκαν στο πρόγραμμα που υλοποιεί τον αλγόριθμο εύρεσης επικαλυπτόμενων κοινοτήτων. Αρχικά, παρουσιάζονται οι χρονικές μεταβολές πριν και μετά την εφαρμογή μεθόδων παραλληλοποίησης. Τέλος, παρουσιάζεται αναλυτικά η εκτέλεση του υλοποιημένου αλγορίθμου σε δύο γράφους δείγματα, σε βήματα.

Το 5ο Κεφάλαιο συνοψίζει τα συμπεράσματα που προέκυψαν κατά την ανάπτυξη του προγράμματος, αναφέρει τις τυχόν βελτιώσεις που μπορούν να γίνουν στην τελική υλοποίηση αλλά και τους μελλοντικούς στόχους για αυτήν.

Κεφάλαιο 2ο Υπόβαθρο

2.1 Εισαγωγή

Η εύρεση κοινοτήτων σε ένα γράφο επιτυγχάνεται με τη χρήση κατάλληλων αλγορίθμων, αποτέλεσμα των οποίων είναι ένα σύνολο από κοινότητες, οι οποίες είναι στην πραγματικότητα υπό-γραφήματα του αρχικού γράφου. Γενικά, ο εντοπισμός κοινοτήτων αποτελεί συνήθως ένα αρκετά βαρύ υπολογιστικά πρόβλημα για τους ηλεκτρονικούς υπολογιστές. Πολύ συχνά, ο αριθμός των κοινοτήτων, αν αυτές υπάρχουν και μπορούν να οριστούν σαφώς στο γράφημα, είναι άγνωστος για τον υπολογιστή. Ο αριθμός των κόμβων που απαρτίζουν την κάθε κοινότητα επίσης δεν είναι σταθερός για όλες. Επίσης διαφορετική για κάθε κοινότητα είναι η πυκνότητα της, δηλαδή ο λόγος των υπαρκτών ακμών μέσα σε αυτήν προς τον μέγιστο αριθμό ακμών που θα μπορούσαν να υπάρξουν σε αυτήν. Παρά τα παραπάνω προβλήματα που συντελούν στην αύξηση υπολογιστικής δυσκολίας του προβλήματος, έχουν αναπτυχθεί διάφοροι αλγόριθμοι, ο κάθε ένας με την δική του αποδοτικότητα και αποτελεσματικότητα. Παρακάτω θα αναφερθούμε σε μερικούς από αυτούς.

2.2 Αλγόριθμος των Girvan Newman

2.2.1 Ανάλυση Αλγόριθμου

Ένας από τους γνωστούς αλγόριθμους εύρεσης κοινοτήτων είναι αυτός που αναπτύχθηκε από τους αναλυτές Michelle Girvan και Mark Newman και είναι μια ιεραρχική μέθοδος εντοπισμού κοινοτήτων σε έναν γράφο. Παρακάτω θα περιγράψουμε την λειτουργία του αλγορίθμου όπως έχει πρωτοεμφανιστεί στην εργασία [4] των παραπάνω αναλυτών. Σε αυτό το σημείο αξίζει να αναφερθούμε σε μια ιδιότητα που παρατηρείται στα περισσότερα δίκτυα του πραγματικού κόσμου, την ιδιότητα μιας κοινότητας στην οποία μερικοί από τους κόμβους ενός γράφου είναι σφιχτά συνδεδεμένοι μεταξύ τους σε ομάδες, μεταξύ των οποίων ομάδων υπάρχουν πιο χαλαρές συνδέσεις. Για παράδειγμα, σε ένα κοινωνικό δίκτυο, όπου οι κόμβοι αναπαριστούν τους χρήστες και οι ακμές την σχέση φιλίας μεταξύ τους, είναι προφανές ότι ενώ όλοι τελικά ανήκουν σε ένα μεγάλο δίκτυο, σίγουρα όμως θα παρατηρείται ότι υπάρχουν και μπορούν να εντοπιστούν συγκεκριμένες ομάδες στις οποίες οι συνδέσεις είναι αρκετά πυκνότερες. Αυτές οι ομάδες θα μπορούσαν να αντιπροσωπεύουν ομάδες ατόμων που συναναστρέφονται συχνά, με κοινά ενδιαφέροντα ή και επαγγελματικό προσανατολισμό. Η δυνατότητα να εντοπιστούν αυτές οι ομάδες-κοινότητες βοηθάει στην βαθύτερη κατανόηση του συγκεκριμένου δικτύου αλλά και στην μετάφραση του δικτύου σε πληροφορία.

Σε αυτό το σημείο αξίζει να αναφερθεί ο τρόπος με τον οποίο ο αλγόριθμος Girvan-Newman καθορίζει το πως ανιχνεύονται οι ομάδες του γράφου. Αντί να οριστεί ένα μέτρο που μας πληροφορεί για τις ακμές οι οποίες είναι κεντρικότερες μέσα στις κοινότητες, ο αλγόριθμος επικεντρώνεται στις ακμές που είναι πιθανότερο να είναι οι ενδιάμεσες των κοινοτήτων. Η

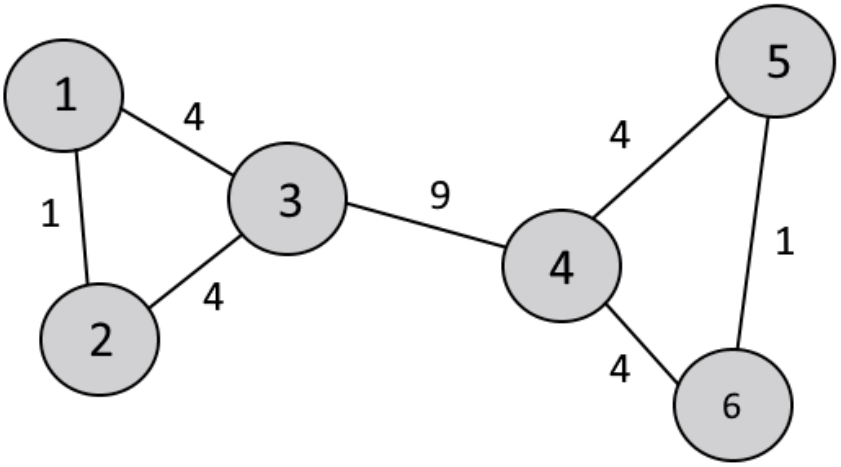
κεντρικότητα κορυφής (Vertex Betweenness) είναι ένα μέτρο κεντρικότητας(centrality) των κορυφών και ως συνέπεια υποδηλώνει την επιρροή τους στον γράφο. Η κεντρικότητα λοιπόν, έχει οριστεί, για μια κορυφή i ως ο αριθμός των συντομότερων μονοπατιών μεταξύ ζευγαριών άλλων κορυφών που όμως διαπερνάνε. Οι αναλυτές στον συγκεκριμένο αλγόριθμο, προεκτείνουν τον προηγούμενο ορισμό στις ακμές και ορίζουν την κεντρικότητα ακμής ως τον αριθμό των συντομότερων διαδρομών μεταξύ ζευγαριών κορυφών που την περιέχουν. Αν υπάρχουν περισσότερα από ένα συντομότερα μονοπάτια μεταξύ δύο κόμβων, δίνεται ίσο βάρος ώστε το συνολικό βάρος όλων των μονοπατιών να είναι ίσο. Αν ένας γράφος περιέχει κοινότητες ή ομάδες που είναι χαλαρά συνδεδεμένες μέσω μερικών, ελάχιστων και εσωτερικών ακμών, τότε όλα τα συντομότερα μονοπάτια μεταξύ των διαφορετικών κοινοτήτων θα περνάνε από μία από αυτές τις ελάχιστες ακμές. Με αυτό τον τρόπο οι ακμές που ενώνουν κοινότητες θα έχουν υψηλή κεντρικότητα ακμής. Αφαιρώντας λοιπόν αυτές τις ακμές με υψηλή κεντρικότητα, οι περιεχόμενες κοινότητες διαχωρίζονται η μία από την άλλη και έτσι αποκαλύπτεται η υπάρχουσα δομή των κοινοτήτων στον γράφο. Ο αλγόριθμος των Girvan-Newman συνοψίζεται στα παρακάτω βήματα:

I. Αρχικά υπολογίζεται η κεντρικότητα όλων των υπαρχόντων ακμών του γράφου.
II. Οι ακμές με την υψηλότερη κεντρικότητα αφαιρούνται.
III. Η κεντρικότητα όλων των ακμών που επηρεάστηκαν από την αφαίρεση των προηγούμενων ακμών υπολογίζεται ξανά.
IV. Τα βήματα II και III επαναλαμβάνονται μέχρι να μην υπάρχουν πλέον ακμές.

Ενδεικτικά, ορισμένες υλοποιήσεις του αλγόριθμου των Girvan-Newman δέχονται ως παράμετρο τον αριθμό επιθυμητών κοινοτήτων ώστε να σταματήσει η επανάληψη όταν πλέον ο αριθμός των εντοπισμένων κοινοτήτων ισούται με αυτόν. Το γεγονός ότι μόνο οι κεντρικότητες που υπολογίζονται ξανά είναι αυτές οι οποίες επηρεάστηκαν από την αφαίρεση που πραγματοποιήθηκε μπορεί να μειώσει τον χρόνο εκτέλεσης άρα και την αποδοτικότητα του αλγορίθμου. Παρόλα αυτά, είναι απαραίτητος ο υπολογισμός της κεντρικότητας σε κάθε βήμα για να μην υπάρξουν προβλήματα. Ο λόγος είναι ότι ο γράφος και οι συνδέσεις του κάθε φορά επηρεάζονται και προσαρμόζονται στην νέα πραγματικότητα που προέκυψε μετά από την αφαίρεση των ακμών. Για παράδειγμα, στην περίπτωση που δύο κοινότητες ενώνονται από πάνω από μία ακμή, δεν είναι σίγουρο ότι όλες τους θα έχουν υψηλή κεντρικότητα. Έτσι, υπολογίζοντας την κεντρικότητα μετά από κάθε αφαίρεση, εξασφαλίζουμε ότι τουλάχιστον μια από τις υπολειπόμενες ακμές θα είναι υψηλές σε κεντρικότητα.

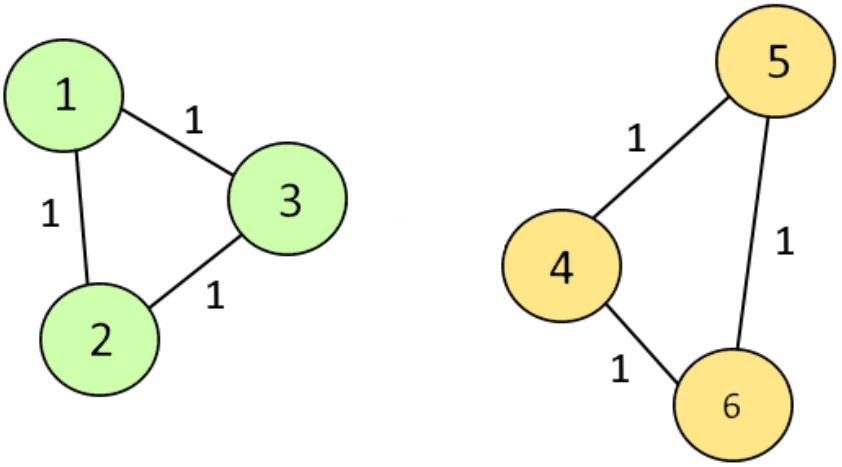
2.2.2 Παράδειγμα Εύρεσης Κοινοτήτων με Girvan-Newman

Το αποτέλεσμα του αλγόριθμου των Girvan-Newman είναι ένα δενδρόγραμμα που παράγεται από πάνω προς τα κάτω όσο ο γράφος διαχωρίζεται σε διαφορετικές κοινότητες με τις διαδοχικές αφαιρέσεις ακμών. Τα φύλλα του δενδρογράμματος είναι κορυφές του αρχικού γράφου. Στις παρακάτω εικόνες φαίνεται η εφαρμογή του αλγόριθμου σε ένα απλό γράφημα. Τα βάρη που έχουν εκχωρηθεί στις ακμές είναι η κεντρικότητα ακμής υπολογισμένη με βάση τον αριθμό συντομότερων διαδρομών που περνούν από αυτές.



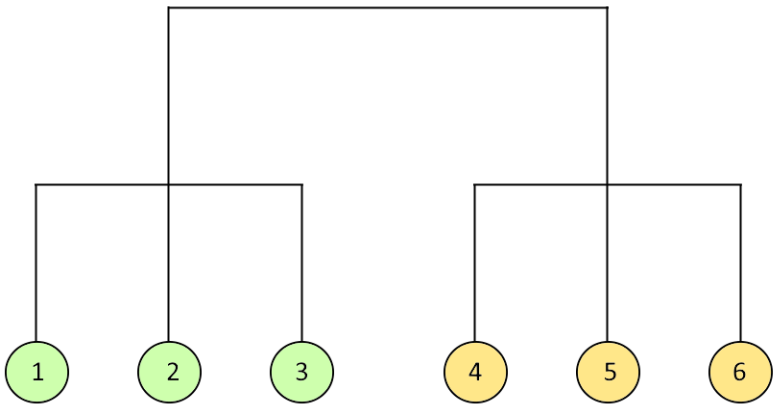
Εικόνα 4: Γράφος Παράδειγμα

Βλέπουμε ότι, στην αρχική κατάσταση, η ακμή με την υψηλότερη κεντρικότητα ίση με 9 είναι αυτή που θα αφαιρεθεί στο πρώτο βήμα εκτέλεσης.



Εικόνα 5: Εφαρμογή Αλγόριθμου Girvan Newman

Στην τελική κατάσταση, βλέπουμε ότι η κεντρικότερη ακμή έχει αφαιρεθεί και έχουν προκύψει οι κοινότητες A = { 1, 2, 3 } και B = { 4, 5, 6 }. Παράλληλα, παρατηρούμε ότι έχουν υπολογιστεί ξανά οι κεντρικότητες κάθε ακμής που επηρεάστηκε από την αφαίρεση αλλά και ότι προκύπτει πως πλέον οι ακμές που απέμειναν είναι ίσης κεντρικότητας μεταξύ τους, άρα δεν υπάρχει λόγος αφαίρεσης περισσοτέρων, αφού σε αυτή την περίπτωση δεν θα απέμεναν άλλες ακμές. Στην επόμενη εικόνα, παρουσιάζεται το τελικό δενδρόγραμμα που παράγεται από τον αλγόριθμο.



Εικόνα 6: Δενδρόγραμμα Εξόδου Αλγόριθμου Girvan-Newman

2.3 Modularity - Τμηματικότητα

2.3.1 Ανάλυση Αλγορίθμου

Όπως έχει αναφερθεί, πολλά ζητήματα επιστημονικού ενδιαφέροντος μπορούν να αναπαρασταθούν σε μορφή γράφων στους οποίους υπάρχει κίνητρο για τον εντοπισμό κοινοτήτων. Όπως έχει οριστεί από τον Michelle Girvan στην δημοσίευση [5], η τμηματικότητα είναι ένα μέτρο που όταν μεγιστοποιείται, μας αποκαλύπτει την δομή των κοινοτήτων ενός γράφου.

Έστω ένας γράφος που περιέχει τμήματα στα οποία παρατηρείται ότι υπάρχουν πολύ πυκνές ενώσεις μεταξύ των κορυφών τους σε σύγκριση με πολύ αραιότερες ενώσεις με τις κορυφές των άλλων τμημάτων. Η τμηματικότητα (modularity) έρχεται να περιγράψει την παραπάνω κατάσταση των τμημάτων αυτών και έτσι ορίζεται ότι αυτά τα τμήματα έχουν υψηλή τμηματικότητα. Συνεπώς, αν μπορούσαμε να υπολογίσουμε την τμηματικότητα διαφόρων τμημάτων του γράφου, θα μπορούσαμε, επιλέγοντας τα τμήματα με τις υψηλότερες τμηματικότιτητες να εντοπίσουμε τις κοινότητες ενός γράφου. Υπάρχουν πολλοί σαφείς τρόποι για να υπολογιστεί η τμηματικότητα, αλλά ο ποιο γνωστός είναι αυτός που ακολουθεί παρακάτω.

Δοθέντος ενός γράφου g με n κορυφές και m ακμές, ώστε ο γράφος να χωρίζεται σε δύο κοινότητες μέσω μιας μεταβλητής s που καθορίζει τα μέλη τους. Αν μια τυχαία κορυφή v ανήκει στην πρώτη κοινότητα, τότε $s_v = 1$ ενώ αν ανήκει στην δεύτερη κοινότητα τότε $s_v = -1$. Έστω ο πίνακας γειτνίασης του γράφου A όπου $A_{vu} = 0$ σηματοδοτεί ότι δεν υπάρχει ακμή μεταξύ των κορυφών v και u και $A_{vu} = 1$ σηματοδοτεί ότι υπάρχει ακμή που να τις ενώνει. Έτσι, $A_{vu} = A_{uv}$. Είναι πιθανό να υπάρχουν περισσότερες από μια ακμές από την κορυφή v στην u αλλά παρουσιάζεται μια απλή κατάσταση για λόγους κατανόησης. Η τμηματικότητα λοιπόν ορίζεται ως Q και ισούται με τον αριθμό των ακμών που βρίσκονται μεταξύ των κοινοτήτων 1 ή 2, μείον των αναμενόμενων ακμών μεταξύ των κοινοτήτων 1 και 2 σε ένα τυχαίο γράφο με την ίδια κατανομή βαθμών για τους κόμβους με τον αρχικό γράφο.

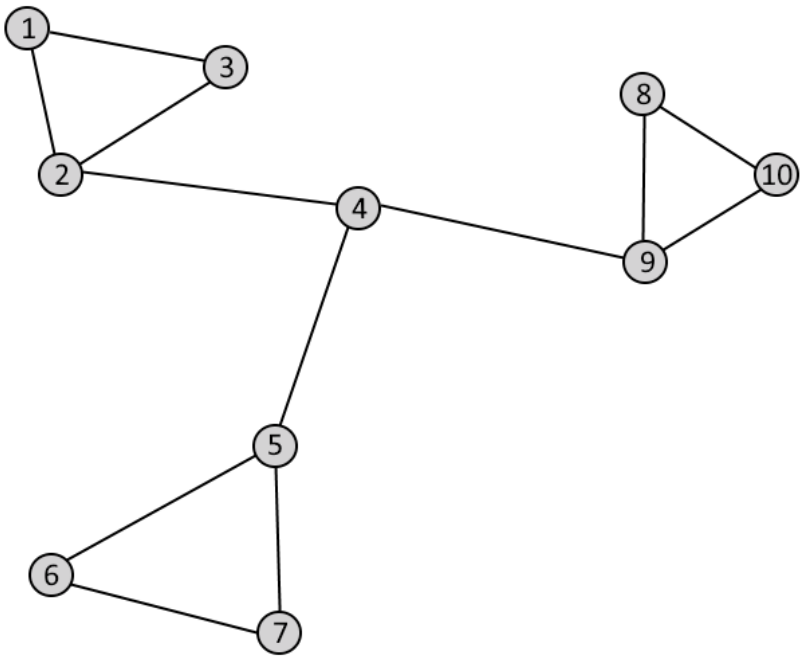
Ο αναμενόμενος αριθμός ακμών ανάμεσα στις κοινότητες θα πρέπει να υπολογιστεί με τη χρήση μιας τυχαιοποιημένης πραγματικότητας του αρχικού γράφου. Συγκεκριμένα, σε έναν γράφο με n κορυφές, όπου κάθε κόμβος v έχει βαθμό k_v , η κάθε ακμή κόβεται σε δύο τμήματα όπου κάθε ένα από αυτά ενώνεται τυχαία με οποιοδήποτε άλλο τμήμα ακμής στον νέο γράφο, επιτρέποντας ακόμα και κύκλους ή πολλαπλά μονοπάτια μεταξύ δύο ίδιων κορυφών. Τελικά, παρά το γεγονός που ο βαθμός του κάθε κόμβου του γράφου παραμένει ο ίδιος, το αποτέλεσμα

της παραπάνω εφαρμογής έχει προκαλέσει την δημιουργία ενός τελείως διαφορετικού, τυχαίου γράφου.

Έστω λοιπόν m ο αναμενόμενος αριθμός ακμών μεταξύ u και v κόμβων με βαθμό k_v και k_u αντίστοιχα σε τυχαίο γράφο. Ορίζεται λοιπόν, ότι για μια κοινότητα C, ο ορισμός της τμηματικότητα, που υποδηλώνει την ισχύ της, είναι το παρακάτω άθροισμα για κάθε ζεύγος της κοινότητας: $\sum_{v\epsilon C,\, u\epsilon C} [A_{vu} - k_v k_u / (2m - 1)]$.

2.3.2 Παράδειγμα Εύρεσης Κοινοτήτων με Modularity

Παρακάτω ακολουθεί ένα συνολικό παράδειγμα για την τμηματικότητα, στο οποίο παρουσιάζεται ένας γράφος με m ακμές στον οποίο υπολογίζεται ο πίνακα γειτνίασης, έτσι ώστε για κόμβους v και u να ισχύει ότι $A_{vu} = 0$ όταν δεν ενώνονται με ακμή και $A_{vu} = 1$ στην αντίθετη περίπτωση. Επιπλέον παρουσιάζουμε τον βαθμό της κάθε κορυφής του γράφου.



Εικόνα 7: Γράφος Παράδειγμα

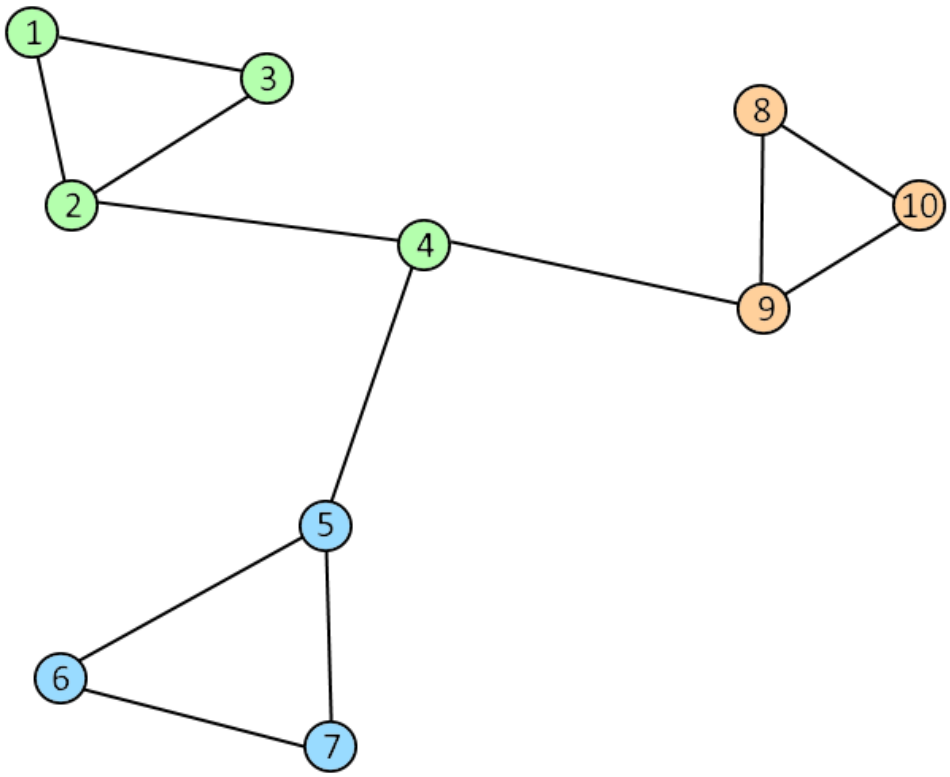
N	1	2	3	4	5	6	7	8	9	10
1	0	1	1	0	0	0	0	0	0	0
2	1	0	1	1	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	0	0
4	0	1	0	0	1	0	0	0	1	0
5	0	0	0	1	0	1	1	0	0	0
6	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	1	1	0	0	0	0
8	0	0	0	0	0	0	0	0	1	1
9	0	0	0	1	0	0	0	1	0	1
10	0	0	0	0	0	0	0	1	1	0

Πίνακας 1: Πίνακας Γειτνίασης Παραδείγματος

Κόμβος	Βαθμός
1	2
2	3
3	2
4	3
5	3
6	2
7	2
8	2
9	3
10	2

Πίνακας 2: Πίνακας Βαθμών Κορυφών

Στην εικόνα που ακολουθεί, παρουσιάζεται ο καλύτερος διαχωρισμός του γράφου που μπορεί να προκύψει μεγιστοποιώντας την τμηματικότητα του, η οποία υπολογίζεται με την βοήθεια των παραπάνω πινάκων.



Εικόνα 8: Εύρεση κοινοτήτων με χρήση Modularity

2.3.3 Ανάλυση Μεθόδου Louvain

Η μέθοδος Louvain είναι μια μέθοδος που αποκαλύπτει τις κοινότητες κορυφών ενός γράφου βασιζόμενη στην τμηματικότητα που παρουσιάστηκε προηγουμένως και δημιουργήθηκε από τον Blondel και άλλους στο πανεπιστήμιο της Λουβαίν του Βελγίου και παρουσιάστηκε στην εργασία [6]. Αποτελεί μια άπληστη μέθοδο βελτιστοποίησης που τρέχει σε χρόνο O(nlogn) όπου n είναι ο αριθμός των κορυφών του γράφου. Ο στόχος δημιουργίας της συγκεκριμένης μεθόδου είναι προφανώς η αποδοτική μεγιστοποίηση της τμηματικότητας (modularity) των πιθανών κοινοτήτων ενός γράφου με απώτερο σκοπό τον θεωρητικά καλύτερο εντοπισμό των κοινοτήτων του.

Όπως εύκολα παρατηρείται, η επανάληψη για διάσχιση όλων των πιθανών ζευγών κορυφών u,v σε έναν γράφο δεν είναι ούτε πρακτική αλλά και ούτε χρονικά αποδοτική, δημιουργείται η ανάγκη για γρηγορότερους ευρετικούς αλγορίθμους. Γενικότερα, η μέθοδος εύρεσης κοινοτήτων του Louvain βρίσκει μικρότερες κοινότητες βελτιστοποιώντας την τμηματικότητα τοπικά για κάθε κορυφή, στην συνέχεια η κάθε μικρή κοινότητα ομαδοποιείται σε μία κορυφή και τέλος επαναλαμβάνει ξανά το πρώτο βήμα. Αυτή η μέθοδος όπως και άλλες, ενώνει κοινότητες των οποίων η συγχώνευση δημιουργεί την μεγαλύτερη πιθανή αύξηση της τμηματικότητας.

Για να μεγιστοποιηθεί η τμηματικότητα αποδοτικά, η μέθοδος Louvain χωρίζεται δύο βήματα τα οποία επαναλαμβάνονται. Αρχικά, η κάθε κορυφή v του γράφου τοποθετείται στην δική του κοινότητα. Στην συνέχεια, για κάθε κορυφή v υπολογίζεται η μεταβολή της τμηματικότητας αφαιρώντας τον v από την κοινότητα του και μετακινώντας τον στην κάθε κοινότητα των γειτονικών κορυφών u του v. Στην συνέχεια, αφού αυτή η τιμή υπολογιστεί για κάθε κοινότητα στην οποία το v είναι συνδεδεμένο στον γράφο, το v τοποθετείτε τελικά στην κοινότητα στην οποία θα προκαλέσει την μεγαλύτερη αύξηση τμηματικότητας. Αν δεν μπορεί να προκύψει αύξηση, το v παραμένει στην αρχική του κοινότητα. Η διαδικασία επαναλαμβάνεται σειριακά σε κάθε κορυφή του γράφου μέχρις ότου να μην μπορεί να προκύψει άλλη αύξηση στην τμηματικότητα. Το πρώτο βήμα της μεθόδου τελειώνει όταν πλέον έχει υπολογιστεί το τοπικό μέγιστο της τμηματικότητας. Στο δεύτερο βήμα, ομαδοποιούνται όλες οι κορυφές στην ίδια κοινότητα και δημιουργείται ένας νέος γράφος όπου οι κορυφές αντιπροσωπεύουν τις κοινότητες του προηγούμενου βήματος. Οι ακμές μεταξύ κόμβων που ανήκουν στην ίδια κοινότητα πλέον απεικονίζονται ως κυκλικές ακμές από την κοινότητα προς την ίδια, ενώ οι ακμές από πολλαπλούς κόμβους μιας κοινότητας σε μια άλλη απεικονίζονται ως ακμές της κοινότητας προς τις άλλες. Όταν πλέον ο νέος γράφος έχει δημιουργηθεί, το δεύτερο βήμα της μεθόδου ολοκληρώνεται και το πρώτο βήμα μπορεί να εφαρμοστεί ξανά σε αυτόν.

2.4 Αλγόριθμος Επικαλυπτόμενων Κοινοτήτων

2.4.1 Περιγραφή Αλγορίθμου

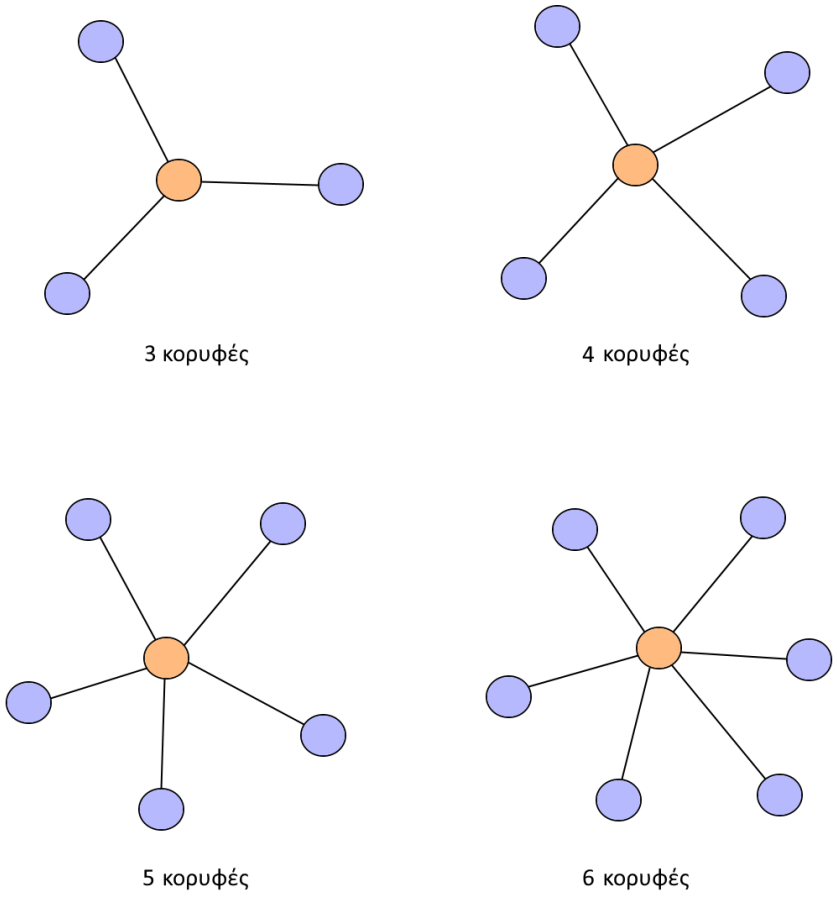
Μια ακόμα από τις μεθόδους εύρεσης κοινοτήτων σε έναν γράφο είναι αυτή της εύρεσης επικαλυπτόμενων κοινοτήτων, που είναι και η μέθοδος που υλοποιήθηκε σε πρακτικό επίπεδο στο πλαίσιο της συγκεκριμένης πτυχιακής εργασίας. Η συγκεκριμένη μέθοδος προσπαθεί να ανακαλύψει τις κοινότητες που υπάρχουν σε έναν γράφο μέσω της εύρεσης των ego networks, η οποίες μπορεί και να επικαλύπτονται μεταξύ τους, δηλαδή μια κορυφή v μπορεί να ανήκει σε δύο διαφορετικές κοινότητες C_1 και C_2. Η πιθανότητα της επικάλυψης δεν είναι καθόλου σπάνια σε γράφους και κοινότητες που απεικονίζουν σκηνές της πραγματικότητας, όπως για παράδειγμα σε ένα δίκτυο μέσου κοινωνικής δικτύωσης, όπου ο κάθε χρήστης ανήκει σε πολλαπλές ομάδες ή και σε ένα δίκτυο βιολογίας που απεικονίζει νευρώνες του εγκεφάλου ενός ζώου που συμβάλλουν σε περισσότερες από μία λειτουργίες.

2.4.2 Ανάλυση Ego Networks

Σε αυτό το σημείο θα αναφερθούμε στο θεωρητικό κομμάτι των ego networks που είναι και το βασικό δομικό συστατικό του αλγόριθμου εύρεσης επικαλυπτόμενων κοινοτήτων. Τα ego networks, συντομογραφικά egonets, όπως περιγράφονται στην εργασία [1] του Linton Freeman είναι κοινωνικά δίκτυα, που απαρτίζονται από κοινωνικές μονάδες, για παράδειγμα προφίλ ατόμων και ομάδες στο διαδίκτυο και των σχέσεων που αναπτύσσονται μεταξύ τους , όπως για παράδειγμα φιλικές και οικογενειακές σχέσεις, που με την σειρά τους συνδέουν ζεύγη μονάδων σε μια ορισμένη δομή. Είναι πάντα χτισμένα γύρο από μια συγκεκριμένη και επιλεγμένη μονάδα που ορίζεται ως ego, δηλαδή το πρόσωπο γύρω από το οποίο σχηματίζεται το egonet. Γενικότερα, για να υπολογίσουμε ένα egonet, επιλέγουμε ένα τμήμα του γράφου όπως μια κορυφή ή και ενδεχομένως ένα μεγαλύτερο σύνολο ως ego, ορίζουμε μια συμμετρική κοινωνική σχέση ή και συνδυασμό σχέσεων και τελικά βρίσκουμε όλα τα υπόλοιπα τμήματα που συνδέονται και συνεπώς σχετίζονται με το ego. Από την παραπάνω διαδικασία εύρεσης ενός egonet, προκύπτουν κοινότητες κορυφών του γράφου που παρατηρείται ότι έχουν ισχυρές σχέσεις και έντονη αλληλεπίδραση μεταξύ τους. Ακολουθεί ο αλγόριθμος υπολογισμού για egonets κορυφών ενός γράφου g:

I. Αρχικά ορίζεται ως ego μια κορυφή veg και η μέγιστη απόσταση αναζήτησης d από αυτή.
II. Στην συνέχεια αναζητούνται στον γράφο τις κορυφές που συνδέονται με το ego σε μέγιστη απόσταση d και σημειώνονται τα ζεύγη που προκύπτουν.
III. Τέλος, η διαδικασία επαναλαμβάνεται για οποιαδήποτε άλλη κορυφή χρειάζεται.

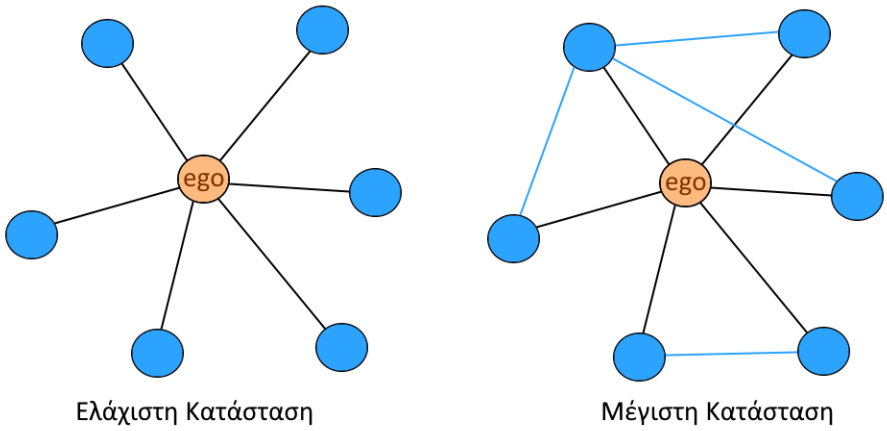
Τα egonets αναπαρίστανται ως γράφοι και συνεπώς παρατηρούνται σε αυτά διάφορες ιδιότητες που ισχύουν και σε κάποιες κατηγορίες γράφων. Ένα egonet είναι ένας γράφος τύπου k-star και έχει ένα συγκεκριμένο κέντρο που είναι προφανώς το ego του. Ένας γράφος τύπου k-star είναι διμερής και έχει ένα κέντρο, ενώ στην ουσία είναι ένα δέντρο το οποίο περιέχει μια εσωτερική κορυφή, k φύλλα και k-1 ακμές. Στην παρακάτω εικόνα απεικονίζονται k-stars με διαφορετικό αριθμό φύλλων.



Εικόνα 9: Παράδειγμα K-star Γράφων

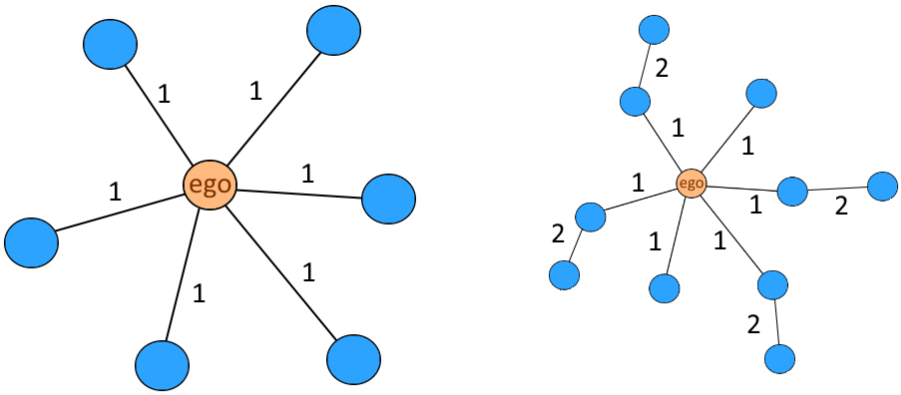
Ως γράφοι με κέντρο (centered graphs), τα egonets είναι συνδεδεμένα και συνεκτικά, δηλαδή υπάρχει πάντα διαδρομή από οποιαδήποτε κορυφή v προς οποιαδήποτε άλλη. Επιπλέον, μια ακόμα ιδιότητα που κληρωνόμουν από τα κεντραρισμένα γραφήματα, είναι ότι το μεγαλύτερο συντομότερο μονοπάτι μεταξύ δυο κόμβων v και u που υπάρχουν σε αυτό ορίζεται ως η διάμετρος του γράφου. Τέλος, για τους γράφους με κέντρο και συνεπώς και για τα ego networks, διακρίνονται δυο ακραίες καταστάσεις. Η πρώτη κατάσταση είναι η ελάχιστη στην οποία το

egonet αναπαρίσταται ως ένα απλό k-star με μόνο τις απαραίτητες ακμές, δηλαδή από το ego προς τις υπόλοιπες κορυφές. Η δεύτερη κατάσταση είναι η μέγιστη, στην οποία παρατηρούμε την συνολική του μορφή, η οποία περιέχει και τις ακμές που είναι πιθανό να υπάρχουν ανάμεσα στα φύλλα του egonet που υποδηλώνουν περαιτέρω σχέσεις μεταξύ αυτών. Οι δύο ακραίες καταστάσεις παρατηρούνται στην παρακάτω εικόνα.



Εικόνα 10: Παρουσίαση Ελάχιστης και Μέγιστης Κατάστασης Egonet

Το τελευταίο στοιχείο που θα μας απασχολήσει είναι η μέγιστη απόσταση αναζήτησης από το ego. Όπως έχει προαναφερθεί, το κάθε πρόβλημα που επιλέγουμε να αναπαραστήσουμε ως γράφο με σκοπό την περαιτέρω μελέτη του, μπορεί να είναι πολύ διαφορετικό από τα υπόλοιπα. Συνεπώς, ανάλογα με την επιλογή του προβλήματος, υπάρχει η πιθανότητα να μας ενδιαφέρει ο συνυπολογισμός στο egonet κορυφών που βρίσκονται παραπάνω από μια θέση «μακριά» από το επιλεγμένο ego. Έτσι λοιπόν, πριν την εκτέλεση της αναζήτησης του egonet, επιλέγεται μια μέγιστη απόσταση αναζήτησης, ώστε να συμπεριληφθούν όλες οι κορυφές για τις οποίες ενδιαφερόμαστε. Η παρακάτω εικόνα περιγράφει μια τέτοια περίπτωση.



Μέγιστη απόσταση ίση με 1 Μέγιστη απόσταση ίση με 2

Εικόνα 11: Επεξήγηση Μέγιστης Απόστασης Αναζήτησης Egonet

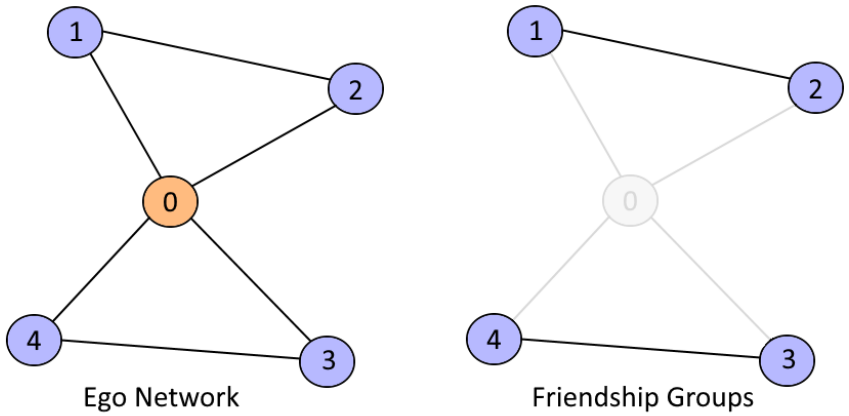
2.5.3 Ανάλυση Αλγορίθμου

Έχοντας αναλύσει τα ego networks φτάνουμε στο αντικείμενο μελέτης της παρούσας πτυχιακής εργασίας που είναι η εύρεση επικαλυπτόμενων κοινοτήτων με την βοήθεια των φιλικών ομάδων που εντοπίζονται για κάθε κόμβο. Η αλγόριθμος ο οποίος υλοποιήθηκε για τους σκοπούς της συγκεκριμένης πτυχιακής εργασίας βασίζεται στην θεωρητική δουλειά [2] των αναλυτών Bradley S. Rees από το Durham University του Ηνωμένου Βασιλείου και Keith B. Gallagher από το Florida Institute of Technology της Μελβούρνης. Όπως έχει προαναφερθεί, στην αναζήτηση επικαλυπτόμενων κοινοτήτων σε έναν γράφο επιτρέπουμε σε μια κορυφή να ανήκει σε περισσότερες από μια κοινότητες.

Δοθέντος ενός προβλήματος, ως υπόβαθρο για την εφαρμογή της μεθόδου, πρέπει προφανώς να προβληθούν τα δεδομένα του προβλήματος ως γράφος με τις σχέσεις αλληλεπίδρασης μεταξύ να απεικονίζονται ως ακμές μεταξύ κορυφών οι οποίες αντιπροσωπεύουν την μικρότερη δυνατή κοινωνική δομή σε αυτόν. Ορίζουμε γενικότερα ότι κοινότητες παρατηρούνται όταν η πυκνότητα των ακμών σε ένα τμήμα του γράφου έχουν μεγαλύτερη πυκνότητα σε αυτό παρά στις συνδέσεις με τα υπόλοιπα τμήματα. Σε αυτό το σημείο, χρησιμοποιούμε την έννοια των ego networks στην μέγιστη ακραία μορφή τους που προαναφέραμε, τα οποία τελικά μας πληροφορούν για όλες τις φιλικές κορυφές μιας οποιαδήποτε κορυφής v στον γράφο. Στην συνέχεια, από τα egonets μπορούμε να εξάγουμε μια νέα δομή, τις φιλικές ομάδες της κάθε κορυφής, η οποίες θα μας βοηθήσουν τελικά στην εύρεση των τελικών κοινοτήτων του γράφου.

Σε αυτό το σημείο, θα γίνει η ανάλυση του συνολικού αλγόριθμου εντοπισμού επικαλυπτόμενων κοινοτήτων όπως έχει παρουσιαστεί από τους αναλυτές. Αρχικά, ορίζουμε τις

φιλικές ομάδες (friendship groups) ως μικρές κοινότητες που εξάγουμε και διαχωρίζουμε από ένα egonet, μιας και αποτελούν μια διαφορετική όψη από αυτό, η οποία οδηγεί τελικά στην αποκάλυψη τον κοινοτήτων που αναζητούμε. Για να εντοπίσουμε μια φιλική ομάδα σε ένα egonet, αφαιρούμε τον κεντρικό ego κόμβο, βρίσκουμε τις συνδεδεμένες κορυφές που απομένουν, από τις οποίες δημιουργούμε λίστες συνιστωσών. Ένα σχετικό παράδειγμα παρατηρείται στην παρακάτω εικόνα, όπου διευκρινίζεται η διαδικασία εξαγωγής των φιλικών ομάδων και η δημιουργία των λιστών συνιστωσών για ένα υποτυπώδες egonet.



Εικόνα 12: Εξαγωγή Φιλικών Ομάδων Egonet

Όπως βλέπουμε λοιπόν για το παραπάνω egonet, μετά την αφαίρεση του κεντρικού κόμβου 0 του egonet προκύπτουν δύο ξεχωριστές συνιστώσες, η πρώτη αποτελείται από τις κορυφές 1 και 2 και την ακμή που τις ενώνει, ενώ η δεύτερη από τις κορυφές 4 και 3 και την ακμή που ενώνει αντίστοιχα. Από αυτές, προκύπτουν δύο λίστες κορυφών, στο τέλος τον οποίων προσθέτουμε και την ego κορυφή που αρχικά αφαιρέσαμε. Έτσι προκύπτουν οι λίστες { 0, 1, 2 } και { 0, 3, 4 }.

Έχοντας ορίσει το βασικό δομικό συστατικό του αλγόριθμου, τις φιλικές ομάδες, μπορούμε να προχωρήσουμε στην παρουσίαση ολόκληρου του αλγόριθμου σε βήματα για εντοπισμό κοινοτήτων σε έναν γράφο g. Ο αλγόριθμος χωρίζεται σε δύο ξεχωριστά τμήματα.

I. Για κάθε κορυφή veg:
   - Υπολογίζεται το egonet του v: H = ego(v).
   - Εξάγονται τα friendship groups από τα egonets.
     - Η ego κορυφή αφαιρείται.
     - Βρίσκονται οι συνδεδεμένες συνιστώσες.
     - Δημιουργούνται οι λίστες συνιστωσών στο τέλος τον οποίον προστίθεται τελικά και ο ego.

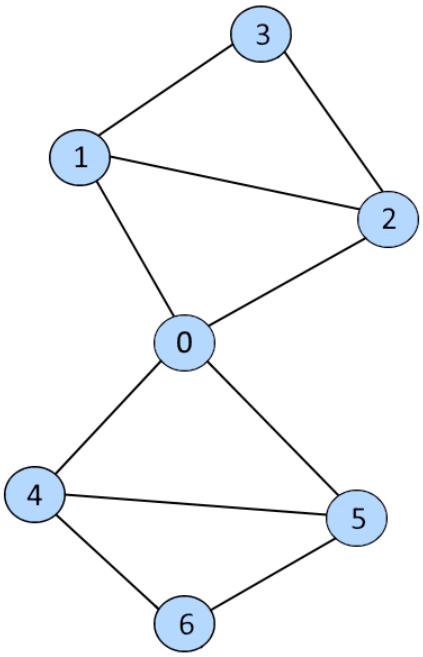
II. Ενώνονται και μειώνονται οι λίστες που προέκυψαν:
   - Αφαιρούνται οι πανομοιότυπες λίστες ή αυτές που είναι γνήσια υποσύνολα άλλων λιστών.
   - Ενώνονται οι λίστες οι οποίες είναι «κοντά» η μια στην άλλη.
   - Η διαδικασία επαναλαμβάνεται μέχρις ότου να μην μπορούν να γίνουν άλλες ενώσεις.

Αναλυτικότερα, στην πρώτη φάση αποκτούμε μια εγωκεντρική άποψη του γράφου, αφού κάθε κορυφή επιλέγεται από μια φορά ως ego με στόχο τον υπολογισμό του egonet και ως συνέπεια των φιλικών ομάδων της. Ο αλγόριθμος διατρέχει λοιπόν την κάθε κορυφή του γράφου για τον συγκεκριμένο σκοπό, επικεντρώνοντας κάθε φορά σε μία διαφορετική κορυφή. Αποτέλεσμα του πρώτης φάσης είναι ουσιαστικά μια συλλογή από λίστες κορυφών, οι οποίες αντιπροσωπεύουν τις τοπικές φιλικές ομάδες που παρατηρούνται στον γράφο και αναπαρίστανται τελικά ως σύνολα κορυφών. Σε αυτό το σημείο, έχει αποκτηθεί γνώση για τους κόμβους που διατηρούν μορφές σχέσεων μεταξύ τους, αλλά η πληροφορία είναι ακόμα αρκετά διάσπαρτη και ακατέργαστη για να προκύψουν συμπεράσματα για τις πραγματικές κοινότητες του γράφου. Δημιουργείται λοιπόν η ανάγκη να ενοποιηθούν οι λίστες φιλικών ομάδων σε μεγαλύτερα σύνολα βήμα προς βήμα, ώστε τελικά να αρχίσουν να σχηματίζονται οι κοινότητες που αναζητούμε.

Στην δεύτερη φάση, γίνεται μια προσπάθεια να ενωθούν οι λίστες φιλικών ομάδων σε μεγαλύτερα σύνολα και τελικά κοινότητες. Αρχικά, απαλείφονται αμέσως από την συλλογή οι φιλικές ομάδες οι οποίες είναι πανομοιότυπες μεταξύ τους. Στην συνέχεια, για μια συλλογή C μεγέθους n, η κάθε λίστα-σύνολο S_1, …, S_n συγκρίνεται με όλες τις υπόλοιπες για να διαπιστωθεί αν είναι γνήσιο υποσύνολο οποιασδήποτε άλλης και στην περίπτωση εντοπισμού ενός τέτοιου συνόλου πραγματοποιείται ένωση. Υπενθυμίζουμε, ότι γνήσιο υποσύνολο ενός συνόλου S_2 καλούμε ένα σύνολο S_1 αν και μόνο αν ισχύει ότι το S_1 είναι υποσύνολο του S_2 και ταυτόχρονα

το S_1 δεν είναι πανομοιότυπο με το S_2. Ακολουθεί το τελευταίο στάδιο ενώσεων, που είναι αυτό της ένωσης συνόλων που δεν ανήκουν στην κατηγορία του πανομοιότυπου συνόλου ή γνήσιου υποσυνόλου αλλά μπορεί να παρατηρηθεί ότι βρίσκονται σχετικά «κοντά» μεταξύ τους. Ορίζεται από τους αναλυτές ότι δύο σύνολα S_1 και S_2 όπου ισχύει ότι $S_1 >= S_2$ θεωρείται ότι βρίσκονται κοντά μεταξύ τους και είναι επιλέξιμα για ένωση αν ισχύει ότι $| S_1 \cup S_2 | = | S_2 | - 1$, δηλαδή αν ο αριθμός τον στοιχείων της ένωσης των δύο συνόλων ισούται με τον αριθμό στοιχείων του δεύτερου μικρότερου συνόλου πλην ένα. Σε αυτή την περίπτωση πραγματοποιείται και πάλι ένωση των συνόλων. Το συγκεκριμένο τελευταίο βήμα πρέπει να επαναληφθεί μέχρις ότου να μην μπορούν πλέον να γίνουν περαιτέρω ενώσεις συνόλων, μιας και έχει ως στόχο να καλύψει το γεγονός ότι ένα egonet δεν έχει πλήρη οπτική του γράφου και των κοινοτήτων.

Για παράδειγμα, στον γράφο της εικόνας που ακολουθεί, προκύπτουν ενδεικτικά τα φιλικά group {1, 2, 3} και {0, 1, 2} τα οποία με βάση την τελευταία ιδιότητα θα ενωθούν σε ένα σύνολο {0, 1, 2, 3, 4}.



Εικόνα 13: Γράφος Παράδειγμα

2.5.4 Απόδοση Αλγόριθμου

Προχωράμε σε αυτό το σημείο στην ανάλυση αποδοτικότητας του αλγορίθμου σύμφωνα με τους αναλυτές. Αναμενόμενα, η απόδοση επηρεάζεται από την πυκνότητα του γράφου που αναλύουμε την κάθε φορά. Για το πρώτο τμήμα του αλγορίθμου, ο εντοπισμός των egonets γίνεται σε σταθερό χρόνο, μιας και δεν απαιτείται καμία τροποποίηση του αρχικού γράφου και το μόνο που απαιτείται είναι να εντοπιστούν οι γείτονες της επιλεγμένης κορυφής την κάθε φορά.

Για την εύρεση των φιλικών ομάδων ή των συνδεδεμένων συνιστωσών, μπορούμε να χρησιμοποιήσουμε αναζήτηση κατά πλάτος Breadth-First Search η οποία τρέχει σε χρόνο O(n+m), όπου n ο αριθμός κορυφών και m ο αριθμός ακμών του egonet. Ο αλγόριθμος κατά πλάτος μπορεί να τροποποιηθεί ώστε να επεξεργάζεται μόνο τις ακμές εντός του egonet οδηγώντας σε χρόνο $O(\log n)^2$. Επιπλέον, η διαδικασία πρέπει να επαναληφθεί, όπως προαναφέρθηκε, για κάθε κορυφή του γράφου. Συνεπώς, ο συνολικός χρόνος εκτέλεσης για το πρώτο τμήμα του αλγορίθμου υπολογίζεται σε $O(n (\log n)^2)$.

Για την ένωση των συνόλων που προκύπτουν από την πρώτη φάση της εκτέλεσης του αλγορίθμου, ώστε να οδηγηθούμε στο αποτέλεσμα των κοινοτήτων που αναζητούμε, μπορεί να χρησιμοποιηθεί ένας τροποποιημένος merge-sort αλγόριθμος. Ένας τυπικός αλγόριθμος merge-sort τρέχει συνήθως σε χρόνο O(n log n), αλλά στην περίπτωση του δικού μας προβλήματος απαιτείται η τροποποίηση του μιας και η ενώσεις μας δημιουργούν νέα σύνολα τα οποία πρέπει να ξανά συγκριθούν με τα εναπομείναντα σύνολα, οδηγώντας σε χρόνο $O(n^2 \log n)$. Αξίζει να αναφερθεί, ότι καθώς η πυκνότητα του γράφου αυξάνεται, τόσο αυξάνεται και η αποδοτικότητα του συγκεκριμένου τμήματος του αλγορίθμου, μιας και όσο η πυκνότητα αυξάνεται, ο αριθμός των πανομοιότυπων ή παρόμοιων συνόλων αυξάνεται επίσης, διευκολύνοντας τον αλγόριθμο και δίνοντάς μας ένα ευκολότερο πρόβλημα προς λύση σε χρόνο O(n). Ο συνολικός χρόνος λοιπόν για ένα αραιό γράφο είναι $O(n(\log n)^2 + n^2 \log n)$ ενώ για αρκετά μεγαλύτερη πυκνότητα τείνει σε $O(n^3 + n)$.

Γενικότερα, ο συγκεκριμένος αλγόριθμος δεν είναι γρηγορότερος σε σύγκριση με άλλους του είδους, αλλά έχει το χαρακτηριστικό ότι μπορεί να βελτιωθεί και να λειτουργήσει σε μια παράλληλη λογική, αυξάνοντας έτσι την αποδοτικότητα αλλά και την επεκτασιμότητα σε μεγαλύτερους γράφους. Το αρχικό στάδιο του αλγορίθμου αποτελείται από την εύρεση των egonets και των friendship groups της κάθε κορυφής και είναι πλήρως παραλληλοποιήσιμο μιας και η διαδικασία για κάθε κορυφή είναι εντελώς ανεξάρτητη από την άλλη. Το δεύτερο στάδιο, αποτελείται από συγκρίσεις ενός επιλεγμένου συνόλου με όλα τα υπόλοιπα κάθε φορά για να αποφανθούμε για την ανάγκη ένωσης, διαγραφής ή διατήρησης με κάποιο σύνολο. Μιας και

κάθε σύγκριση είναι επίσης ανεξάρτητη από τις υπόλοιπες, αυτό το τμήμα του αλγορίθμου μπορεί επίσης να εκτελεστεί παράλληλα.

Συμπερασματικά, οι αναλυτές αναφέρουν ότι εκτός από το γεγονός ότι ο αλγόριθμος παραθέτει μια διαφορετική άποψη για τον εντοπισμό κοινοτήτων συμπεριλαμβανομένης και της επικάλυψης αυτών, διακρίνεται και η δυνατότητα βελτίωσης της αποδοτικότητας μέσω της κατάλληλης βελτιστοποίησης, ακόμα και ίσως μέσω της χρήσης των δυνατοτήτων της μονάδας διαχείρισης γραφικών του υπολογιστή, που προσφέρει νέες και διαφορετικές προοπτικές παραλληλοποίησης του κώδικα.

Κεφάλαιο 3ο Υλοποίηση

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα περιγράψουμε την υλοποίηση του αλγόριθμου επικαλυπτόμενων κοινοτήτων που δημιουργήθηκε για τους σκοπούς της συγκεκριμένης πτυχιακής εργασίας. Η γλώσσα προγραμματισμού που επιλέχθηκε για την συγκεκριμένη υλοποίηση είναι η Java, σε πακέτο ανάπτυξης Java Development Kit έκδοσης 17.0.1, ενώ παράλληλα για την διευκόλυνση της ανάπτυξης χρησιμοποιήθηκε η βιβλιοθήκη χειρισμού γράφων JGraphT.

Αξίζει να κάνουμε μια σύντομη αναφορά στην open source βιβλιοθήκη JGraphT [3] ως προς τις δυνατότητες που προσφέρει σε προγραμματιστές που ασχολούνται με την θεωρία γράφων, τις δομές δεδομένων γενικότερα ή και ακόμα και με την υλοποίηση και ανάπτυξη αλγορίθμων για αυτά. Αρχικά, επιτρέπει στους χρήστες να ορίσουν ως τύπο κορυφής και συνεπώς μονάδα μελέτης οποιοδήποτε τύπο μεταβλητής επιθυμούν, χρησιμοποιώντας generics της Java. Επιπλέον, παρέχει γενικές υλοποιήσεις για βασικούς γράφους ή άλλες δομές δεδομένων αλλά ακόμα και πιο συγκεκριμένες, όπως για παράδειγμα ψευδογράφους, ενώ επιτρέπει και διαφορετικούς τύπους ακμών, δηλαδή κατευθυνόμενες ή μη και με βάρη ή χωρίς. Πέρα των βασικών δομών που προσφέρει, παρέχει και μια μεγάλη λίστα υλοποιημένων αλγορίθμων όπως για παράδειγμα υλοποίηση για τον αλγόριθμο των Girvan-Newman, iterators για διάσχιση των γράφων τύπου Breadth-First ή Depth-First και άλλα. Τέλος, οι συνεισφέροντες έχουν μεριμνήσει ώστε η βιβλιοθήκη να φροντίζει και για την αποδοτικότητα, παρέχοντας σχεδόν native χρόνους απόδοσης σε πολλές περιπτώσεις.

3.2 Περιγραφή Γεννήτριας Egonet

Στην συγκεκριμένη υλοποίηση που αναπτύχθηκε, αποφασίστηκε ο αλγόριθμος να υλοποιηθεί σε δύο τμήματα, ένα το οποίο υπολογίζει γενικότερα τα ego networks μιας κορυφής ενός γράφου και ένα το οποίο εκτελεί τον αλγόριθμο επικαλυπτόμενων κοινοτήτων χρησιμοποιώντας το πρώτο για να υπολογίσει τα egonets της κάθε κορυφής. Έτσι, δημιουργήθηκε μια κλάση γεννήτρια που παράγει το egonet μιας κορυφής ενός γράφου. Η βιβλιοθήκη JGraphT περιλαμβάνει κλάσεις οι οποίες καλούνται γεννήτριες (generators) και γενικότερα χρησιμεύουν στην παραγωγή νέων γράφων ανάλογα με τις ανάγκες του κάθε πειράματος ή αλγορίθμου. Επιπλέον παρέχεται μια γενικότερη κλάση η οποία καλείται GraphGenerator και περιγράφει την δομή μιας γεννήτριας και υλοποιείται ανάλογα με τις ανάγκες του κάθε αναλυτή. Η μορφή της κλάσης γεννητριών είναι:

- Interface GraphGenerator<V, E, T>
  - V - Ο τύπος κορυφών του γράφου.
  - E - Ο τύπος ακμών του γράφου.
  - T - Ο τύπος για επιστροφή ειδικών mappings από String προς στοιχεία του γράφου αν απαιτείται από την υλοποίηση.

Αξίζει να επισημάνουμε ότι στην συγκεκριμένη βιβλιοθήκη επιδιώκεται να επιτρέπεται η χρήση οποιουδήποτε τύπου κορυφών και μιας σειράς τύπων ακμών, οπότε χρησιμοποιούνται generics της Java στους τύπους V, E και T. Απαιτείται επιπλέον η υλοποίηση της μεθόδου που ουσιαστικά παράγει τον ζητούμενο γράφο:

- void generateGraph(Graph<V, E>, target, Map<String, T> resultMap)
  - target - Ο γράφος ο οποίος θα παραχθεί θα τοποθετηθεί σε αυτή την μεταβλητή.
  - resultMap - Αν δεν είναι null, δέχεται τα ειδικά mappings όταν αυτό απαιτείται από την υλοποίηση.

Η κλάση που δημιουργήθηκε για το πρώτο τμήμα του αλγορίθμου είναι μια γεννήτρια, καλείται EgonetGraphGenerator και υλοποιεί την GraphGenerator που προαναφέρθηκε, έχοντας την παρακάτω μορφή:

- Interface EgonetGraphGenerator<V, E, V> implements GraphGenerator<V, E, V>
  - V - Ο τύπος κορυφών του γράφου.
  - E - Ο τύπος ακμών του γράφου.

Επιπλέον ας παρουσιάσουμε τα πεδία της παραπάνω κλάσης:

- private final Graph<V, E> graph
  - Η μεταβλητή στην οποία ο κατασκευαστής (constructor) εκχωρεί το γράφο στον οποίο υπάρχει η κορυφή ego της οποίας ψάχνουμε το egonet.

- private final V ego
  - Η μεταβλητή στην οποία ο κατασκευαστής εκχωρεί την κορυφή ego της οποίας ψάχνουμε το egonet.

- private final int radius
  - Η μεταβλητή στην οποία ο κατασκευαστής εκχωρεί την επιθυμητή μέγιστη απόσταση αναζήτησης από το ego προς τις κορυφές του egonet.

- private final boolean includeEgo
    - Μεταβλητή τύπου flag που καθορίζει αν θα συμπεριληφθεί η κορυφή ego στο τελικό egonet γράφο που θα παραχθεί ή όχι.

- private final boolean newEdges
    - Μεταβλητή τύπου flag που καθορίζει αν θα χρησιμοποιηθούν νέες ακμές στο egonet που θα παραχθεί (νέα αντικείμενα Java) ή αν θα χρησιμοποιηθούν οι υπάρχοντες του αρχικού γράφο.

- public static final int DEFAULT_RADIUS
    - Η προκαθορισμένη τιμή για την μέγιστη απόσταση αναζήτησης για το egonet radius, η οποία ισούται με 1.

- public static final boolean DEFAULT_INCLUDE_EGO
    - Η προκαθορισμένη τιμή για την μεταβλητή includeEgo, που καθορίζει αν θα συμπεριληφθεί η ego κορυφή ή όχι, η οποία ισούται με true.

- public static final boolean DEFAULT_USE_NEW_EDGES
    - Η προκαθορισμένη τιμή για την μεταβλητή newEdges, που καθορίζει αν θα χρησιμοποιηθούν νέες ακμές, η οποία ισούται με true.

Όπως παρατηρείται, υπάρχουν προκαθορισμένες τιμές για τα πλέον εξειδικευμένα πεδία της κλάσης, ώστε να μπορεί να χρησιμοποιηθεί και χωρίς να γίνει ανάθεση σε αυτά στον constructor. Απολύτως απαραίτητα για την δημιουργία ενός αντικειμένου EgonetGraphGenerator και συνεπώς στην εκτέλεση της γεννήτριας είναι μόνο ο γράφος στον οποίο δουλεύουμε και η ego κορυφή. Οι γράφοι που μπορούν να χρησιμοποιηθούν από την συγκεκριμένη γεννήτρια επιτρέπεται να είναι κατευθυνόμενοι ή και όχι.

Κατά την διάρκεια της υλοποίησης, αποφασίστηκε να συμπεριληφθεί η μέγιστη απόσταση αναζήτησης από την ego κορυφή (radius) ώστε να μπορούν να καλυφθούν οι περιπτώσεις πειραμάτων και εφαρμογών στα οποία ενδιαφερόμαστε για κορυφές που βρίσκονται λίγο πιο μακριά από τον ego. Επιπλέον, προστέθηκε η δυνατότητα η γεννήτρια να επιστρέφει στο τέλος μόνο τον υπο-γράφο που είναι συνδεδεμένος με την ego κορυφή χωρίς να περιλαμβάνεται η ίδια, σε περίπτωση που μας ενδιαφέρουν μόνο οι συνδέσεις και οι σχέσεις μεταξύ των φύλλων του egonet. Τέλος, προσφέρεται η δυνατότητα να δημιουργηθούν νέα αντικείμενα Java για τις ακμές του egonet, μιας και σε διάφορα προβλήματα μπορεί να απαιτείται η επεξεργασία του ενώ παράλληλα θα θέλαμε να αποφύγουμε την αλλοίωση των ακμών του αρχικού γράφου.

Σε αυτό το σημείο επικεντρωνόμαστε στον κατασκευαστή (constructor) αντικειμένων της κλάσης EgonetGraphGenerator. Στην γενική του μορφή, δέχεται όλα τα προαναφερθέντα πιθανά πεδία της κλάσης, συμπεριλαμβανομένου της απόστασης radius, και των λογικών μεταβλητών flag includeEgo και newEdges, ενώ στην συντομότερη εκδοχή του απαιτεί ως είσοδο μόνο έναν γράφο και την ego κορυφή. Ως προς την λειτουργία του, ελέγχει ότι η κορυφή ego ως προς την ορθότητα της και σε περίπτωση που αυτή δεν ανήκει στον γράφο που δόθηκε η δημιουργία τερματίζει και εξάγεται εξαίρεση (exception) για λάθος στοιχεία εισόδου. Επιπλέον, ελέγχεται ότι ο γράφος που δόθηκε δεν είναι κενός, και σε περίπτωση που ο ίδιος περιέχει βάρη στις ακμές τα αγνοεί. Τέλος, τα πεδία του νέου αντικειμένου αρχικοποιούνται, είτε με τις δοθείσες τιμές αν κλήθηκε η πλήρης μορφή του κατασκευαστή ή με τις αρχικές (default) που προϋπάρχουν. Οι διάφορες παραλλαγές του αρχικού πλήρους ορισμένου κατασκευαστή πραγματοποιούν κλήση αυτού με την χρήση των αρχικών τιμών για τα πεδία που δεν ορίστηκαν από τον χρήστη. Παρακάτω βλέπουμε πιο συγκεκριμένα τους δυνατούς τρόπους κλήσης του κατασκευαστή.

- public EgonetGraphGenerator(Graph<V, E> graph, V ego, int radius, boolean includeEgo, boolean newEdges)
  - Η πλήρης μορφή κλήσης του κατασκευαστή.

- public EgonetGraphGenerator(Graph<V, E> graph, V ego, int radius, boolean includeEgo)
  - Η μορφή του κατασκευαστή που δεν περιλαμβάνει το πεδίο newEdges για το οποίο θα χρησιμοποιηθεί η προαναφερθείσα τιμή DEFAULT_USE_NEW_EDGES

- public EgonetGraphGenerator(Graph<V, E> graph, V ego, int radius)
  - Η μορφή του κατασκευαστή που δεν περιλαμβάνει ούτε το πεδίο newEdges αλλά ούτε το πεδίο includeEgo, για τα οποία θα χρησιμοποιηθούν οι τιμές DEFAULT_USE_NEW_EDGES και DEFAULT_INCLUDE_EGO αντίστοιχα.

- public EgonetGraphGenerator(Graph<V, E> graph, V ego, boolean includeEgo)
  - Η μορφή του κατασκευαστή που δεν περιλαμβάνει τα πεδία newEdges και radius για τα οποία θα χρησιμοποιηθούν οι τιμές DEFAULT_USE_NEW_EDGES και DEFAULT_RADIUS αντίστοιχα.

- public EgonetGraphGenerator(Graph<V, E> graph, V ego)
    - Η απλούστερη μορφή του κατασκευαστή που απαιτεί μόνο τον γράφο και την ego κορυφή, ενώ χρησιμοποιεί όλες τις αρχικές τιμές DEFAULT_RADIUS, DEFAULT_INCLUDE_EGO και DEFAULT_USE_NEW_EDGES για τα υπόλοιπα πεδία.

Τέλος, ας περιγράψουμε την λειτουργία της συγκεκριμένης γεννήτριας παραγωγής egonet η οποία υλοποιείται στην μέθοδο generateGraph της κλάσης. Ας υπενθυμίσουμε, ότι ένα egonet απαρτίζεται από την ego κορυφή, τις κορυφές γύρω από αυτή στην δοσμένη απόσταση αλλά και από τις σχέσεις-συνδέσεις μεταξύ τους. Στην διαδικασία ανάπτυξης, αποφασίστηκε να χρησιμοποιηθούν αλγόριθμοι αναζήτησης για να βρεθούν αρχικά οι κορυφές που απαρτίζουν το egonet, ενώ στην συνέχεια να βρεθούν μέσω του αρχικού γράφου οι ακμές που τις συνδέουν και πρέπει να συμπεριληφθούν σε αυτό. Σε αυτό το σημείο η λειτουργία διακλαδώνεται στην πρώτη περίπτωση για μη κατευθυνόμενους γράφους και στην δεύτερη για κατευθυνόμενους.

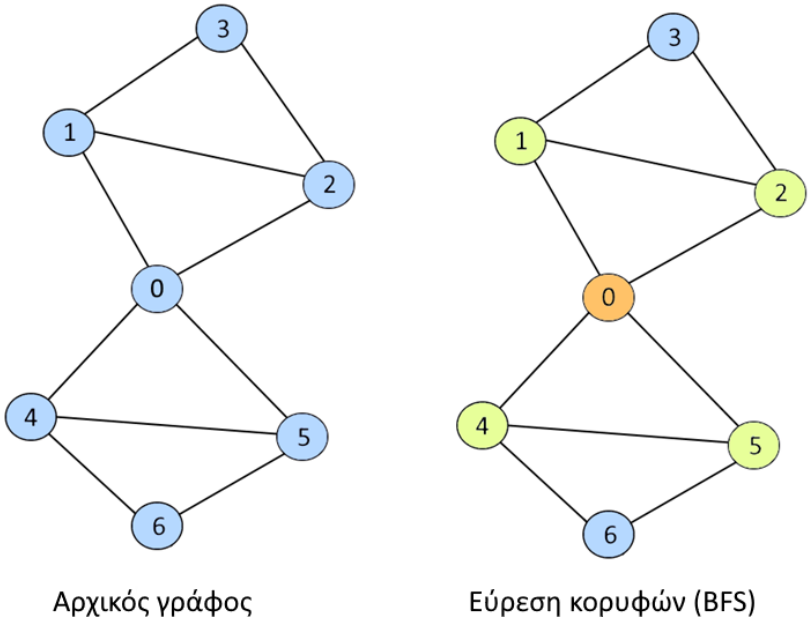
Στην περίπτωση των μη κατευθυνόμενων γράφων, για την εύρεση τον κορυφών του egonet αποφασίστηκε να χρησιμοποιηθεί ο αλγόριθμος αναζήτησης κατά πλάτος (Breadth-First Search). Ο αλγόριθμος αναζήτησης κατά πλάτος, δοθέντος μιας κορυφής-ρίζας ενός δέντρου και κατ' επέκταση γράφου, εκκινεί από την ρίζα και εξερευνά όλες τις συνδεδεμένες σε αυτή κορυφές πρώτού προχωρήσει σε μεγαλύτερο βάθος. Για την υλοποίηση του χρησιμοποιείται συνήθως μια ουρά ώστε να κρατούνται προσωρινά στην μνήμη παιδιά κορυφών που ανακαλύφθηκαν αλλά δεν εξερευνήθηκαν ακόμα λόγω βάθους. Για τις ανάγκες τις δικής μας αναζήτησης σε έναν μη κατευθυνόμενο γράφο, δηλαδή την εύρεση κορυφών του egonet στην ορισμένη μέγιστη απόσταση αναζήτησης που ορίζει ο χρήστης, αρκεί να γίνει μια αναζήτηση κατά πλάτος με βάθος ίσο με την μέγιστη επιθυμητή απόσταση αναζήτησης. Η βιβλιοθήκη JGraphT, παρέχει μια υλοποίηση του αλγορίθμου αναζήτησης κατά βάθος που καλείται BFSShortestPath και δέχεται ως είσοδο την κορυφή ρίζα. Η συγκεκριμένη υλοποίηση της βιβλιοθήκης δεν παρείχε την δυνατότητα τερματισμού της αναζήτησης σε συγκεκριμένο βάθος, οπότε προστέθηκε σε αυτήν το προαιρετικό πεδίο radius που αντιπροσωπεύει την μέγιστη απόσταση αναζήτησης. Σε περίπτωση που αυτή δεν δοθεί από τον χρήστη, θεωρείται ότι ισούται με το θετικό άπειρο ώστε να μην τερματιστεί η αναζήτηση μέχρι να ολοκληρωθεί η εξερεύνηση όλου του γράφου.

Στην τελευταία περίπτωση των κατευθυνόμενων γράφων, για την εύρεση τον κορυφών του egonet χρησιμοποιήθηκε ο αλγόριθμος του Dijkstra. Για την ιστορία, ο συγκεκριμένος αλγόριθμος αναπτύχθηκε για πρώτη φορά από τον επιστήμονα της επιστήμης της πληροφορικής Edsger W. Dijkstra το 1956 και δημοσιεύτηκε λίγο αργότερα. Είναι ένας από τους πιο γνωστούς αλγόριθμους αναζήτησης σε δίκτυα, ικανός να βρίσκει τα συντομότερα μονοπάτια

μεταξύ κορυφών ενός γράφου. Δοθέντος ενός ζεύγους κορυφών, ο αλγόριθμος του Dijkstra επιλέγει το κοντινότερο σε απόσταση κόμβου που δεν έχει ξανά επισκεφτεί, υπολογίζει την απόσταση μέσω αυτού προς κάθε κόμβο που δεν έχει ξανά επισκεφτεί, και ανανεώνει την απόσταση των γειτόνων αν η νεότερη είναι μικρότερη από την προ υπάρχουσα. Συνήθως στην υλοποίηση χρησιμοποιείται μια ουρά προτεραιότητας. Η βιβλιοθήκη JGraphT παρέχει μια υλοποίηση του αλγόριθμου του Dijkstra που καλείται DijkstraShortestPath και δέχεται ως είσοδο την κορυφή ρίζα και μία μέγιστη απόσταση αναζήτησης, και βρίσκει όλα τα συντομότερα μονοπάτια σε αυτήν. Σε αυτή την περίπτωση δεν χρειάστηκε να προσθέσουμε την λειτουργία απόστασης μιας και αυτή προϋπήρχε στην αρχική υλοποίηση.

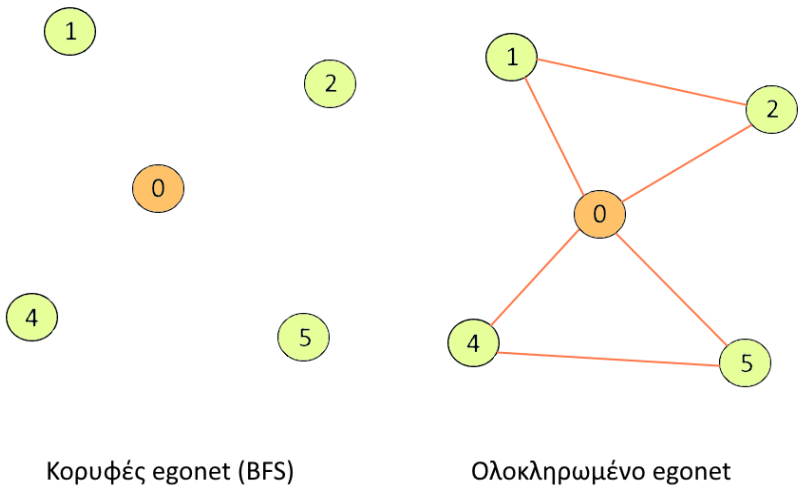
Όμοια, όποιος αλγόριθμος από τους παραπάνω να χρησιμοποιηθεί, επιστρέφεται ένας χάρτης τύπου searchMap που απεικονίζει τις συντομότερες διαδρομές και έχει ως key set τις κορυφές που χρειαζόμαστε και εκχωρούμε στο γράφο εξόδου target ώστε να περάσουμε στην επόμενη φάση εκτέλεσης της γεννήτριας. Σε αυτό το σημείο, ανάλογα με την τιμή που όρισε ο χρήστης για την μεταβλητή includeEgo κατά την δημιουργία του αντικειμένου της γεννήτριας, αφήνεται ή αφαιρείται η κορυφή ego του. Στην δεύτερη φάση εκτέλεσης, διατρέχονται οι κορυφές που βρήκαμε προηγουμένως και αναζητούνται στον αρχικό γράφο όλες οι ακμές που υπήρχαν μεταξύ τους. Αφού αυτές βρεθούν, ανάλογα με την τιμή που όρισε ο χρήστης για την μεταβλητή newEdges κατά την δημιουργία του αντικειμένου της γεννήτριας, χρησιμοποιούνται και προστίθενται στον γράφο στόχο οι ακμές είτε ως νέα ή υπάρχοντα αντικείμενα ακμών.

Στην παρακάτω εικόνα παρατηρούμε την διαδικασία εύρεσης του egonet όπως παρουσιάστηκε για έναν μη κατευθυνόμενο γράφο.



Εικόνα 14: Εύρεση Egonet Μέρος Πρώτο

Όπως φαίνεται στην παραπάνω εικόνα, έχουμε έναν μη κατευθυνόμενο γράφο και αναζητούμε το egonet της κορυφής 0 σε αυτόν με μέγιστη απόσταση αναζήτησης ίση με ένα. Εκτελούμε αναζήτηση BFS με ρίζα την κορυφή μηδέν, και προκύπτει το σύνολο κορυφών Σ = {0, 1, 2, 4, 5}. Επιθυμούμε να διατηρήσουμε την κορυφή ego οπότε δεν την αφαιρούμε από το σύνολο. Δημιουργούμε έναν νέο γράφο, που θα αποτελεί το egonet εξόδου με σύνολο κορυφών το Σ. Στο πρώτο μέρος της παρακάτω συμπληρωματική εικόνα παρατηρούμε το egonet που έχει βρεθεί. Στην συνέχεια διατρέχουμε τις κορυφές του συνόλου Σ στον αρχικό γράφο και αντιγράφουμε της ακμές που υπάρχουν μεταξύ τους στο egonet που δημιουργήσαμε προηγουμένως. Ως αποτέλεσμα, έχουμε πλέον βρει ολόκληρο το egonet της κορυφής 0 στον αρχικό γράφο.



Εικόνα 15: Εύρεση Egonet Μέρος Δεύτερο

3.3 Περιγραφή Υλοποίησης Αλγορίθμου

Στο δεύτερο τμήμα της υλοποίησης παίρνει μορφή ο κεντρικός αλγόριθμος εύρεσης επικαλυπτόμενων κοινοτήτων που αντιπροσωπεύεται από την δική του κλάση Java και χρησιμοποιεί την γεννήτρια που ορίστηκε πριν για τα egonets κορυφών. Παράλληλα, στην συγκεκριμένη κλάση έχουν χρησιμοποιηθεί στοιχεία παραλληλοποίησης σε συγκεκριμένα σημεία με στόχο την βελτίωση του χρόνου εκτέλεσης σε μεγαλύτερους γράφους που επιβαρύνουν αρκετά την εκτέλεση. Η βιβλιοθήκη JGraphT παρέχει και σε αυτή την περίπτωση γενικές κλάσεις προς υλοποίηση για την περιγραφή διαφόρων αλγορίθμων. Στην συγκεκριμένη περίπτωση, χρησιμοποιήθηκε και υλοποιήθηκε η γενική κλάση της JGraphT που καλείται ClusteringAlgorithm και έχει ως σκοπό την περιγραφή αλγορίθμων εύρεσης συστάδων και κοινοτήτων. Η μορφή της κλάσης είναι:

- public interface ClusteringAlgorithm<V>
  - V – Ο τύπος κορυφών του γράφου.

- public Clustering<V> getClustering()

Η κλάση που δημιουργήσαμε λοιπόν καλείται OverlappingClustering και ορίζεται παρακάτω:

- public class OverlappingClustering<V, E> implements ClusteringAlgorithm<V>
  - V – Ο τύπος κορυφών του γράφου.
  - E – Ο τύπος ακμών του γράφου.

Επιπλέον ας παρουσιάσουμε τα πεδία της παραπάνω κλάσης:

- private final Graph<V, E> graph
  - Η μεταβλητή στην οποία ο κατασκευαστής (constructor) εκχωρεί τον γράφο στον οποίο θα εκτελεστεί η εύρεση κοινοτήτων.

- private final int parallelism
  - Η μεταβλητή που καθορίζει το μέγεθος παραλληλοποίησης.

- private final ThreadPoolExecutor executor
  - Η μεταβλητή η οποία διατηρεί τον executor των threads που θα δημιουργηθούν.

- private final int radius
  - Η μεταβλητή στην οποία ο κατασκευαστής εκχωρεί την επιθυμητή μέγιστη απόσταση αναζήτησης για τα egonets που θα αναζητηθούν

- private final List<Set<V>> friendshipGroups
  - Η μεταβλητή που θα συλλέξει τις φιλικές ομάδες που θα παράγονται.

- private final List<Set<V>> superGroups
  - Η μεταβλητή που θα συλλέξει τις φιλικές ομάδες που πλέον έχουν περάσει από την διαδικασία αφαίρεσης των γνήσιων υποσυνόλων.

- private final List<Set<V>> mergedGroups
  - Η μεταβλητή που θα συλλέξει τις ομάδες που πλέον έχουν περάσει από το τελικό στάδιο ενώσεων ανάλογα με το ποσό «κοντά» βρίσκονται με τις υπόλοιπες και τελικά θα αποτελέσουν τις κοινότητες που αναζητούμε.

- public static final int DEFAULT_PARALLELISM
  - Η προκαθορισμένη τιμή παραλληλοποίησης ορίζεται ως ο μέγιστος αριθμός νημάτων που διαθέτει η συγκεκριμένη συσκευή προς χρήση.

- public static final int DEFAULT_RADIUS
  - Η προκαθορισμένη τιμή για την μέγιστη απόσταση αναζήτησης στα egonet που θα βρεθούν.

Όμοια με την προηγούμενη κλάση που περιγράψαμε, υπάρχουν προκαθορισμένες τιμές για τα εξειδικευμένα πεδία της κλάσης ώστε να μπορούν να παραλειφθούν από την κλήση του κατασκευαστή, αν δεν εξυπηρετούν κάποιον χρήστη. Ως προκαθορισμένη τιμή για τον αριθμό παραλληλοποίησης ορίζουμε τον μέγιστο αριθμό νημάτων που μπορεί να μας διαθέσει το runtime της Java σε κάθε δημιουργία αντικειμένου. Απολύτως απαραίτητο για την δημιουργία του αντικειμένου του αλγορίθμου είναι μόνο ο γράφος στον οποίο θα γίνει η εύρεση κοινοτήτων.

Επιπλέον, στα πρότυπα της βιβλιοθήκης JGraphT, η οποία περιέχει κλάσεις δοκιμών της μορφής Test του Junit για την κλάση παραγωγής των ego networks αλλά και για την κλάση που υλοποιεί τον αλγόριθμο εύρεσης επικαλυπτόμενων κοινοτήτων. Στην περίπτωση των egonets περιέχονται δοκιμές που αποδεικνύουν ότι η γεννήτρια μπορεί να διαχειριστεί κατευθυνόμενους, μη κατευθυνόμενους αλλά και γράφους με βάρος, ενώ παράλληλα δοκιμάζεται η χρήση και η σωστή λειτουργία όλων των παραμέτρων της κλάσης. Όσον αφορά τις δοκιμές για την κλάση του αλγορίθμου, περιλαμβάνονται δοκιμές για κάθε ξεχωριστό τμήμα του αλγορίθμου αλλά και ένα συνολικό. Η μορφή των κλάσεων δοκιμών είναι η παρακάτω:

- public class EgonetGraphGeneratorTest

- public class OverlappingClusteringTest

Όπως προαναφέρθηκε, αποφασίστηκε συγκεκριμένα τμήματα του αλγορίθμου να παραλληλοποιηθούν όταν δεν παρατηρείται αλλοίωση του αποτελέσματος εξόδου για να επιτευχθούν καλύτεροι χρόνοι εκτέλεσης, ειδικότερα σε πυκνότερους και μεγαλύτερους γράφους. Έτσι ορίζονται τα πεδία parallelism και executor της κλάσης ώστε να μπορέσει να πραγματοποιηθεί στην συνέχεια στο πρακτικό τμήμα εκτέλεσης η χρήση νημάτων. Επιπλέον, υπάρχει η ανάγκη ορισμού πεδίων-λιστών οι οποίες θα διατηρούν τις ομάδες (friendshipGroups, superGroups, mergedGroups) εσωτερικά της κλάσης, ως υπόβαθρο για την χρήση νημάτων, τα οποία στην συνέχεια θα γεμίζουν σιγά-σιγά τις συγκεκριμένες λίστες.

Επικεντρωνόμαστε και πάλι στον κατασκευαστή (constructor) αντικειμένων της κλάσης εύρεσης κοινοτήτων. Στην γενική του πλήρη μορφή δέχεται όλα τα προαναφερθέντα πιθανά πεδία της κλάσης, δηλαδή τον γράφο στον οποίο θα γίνει η αναζήτηση, τον executor των νημάτων, και της μέγιστης απόστασης των egonet, ενώ στην συντομότερη εκδοχή απαιτείται ως παράμετρος μόνο ο γράφος. Ως προς την λειτουργία του, πέρα από τις εκχωρήσεις, ελέγχει αν ο γράφος είναι κενός, κάτι που αν ισχύει οδηγεί στον τερματισμό λειτουργίας και την εξαγωγή εξαίρεσης. Οι διάφορες παραλλαγές του κατασκευαστή που προσφέρονται χρησιμοποιούν κλήση προς την πλήρη του μορφή χρησιμοποιώντας της προκαθορισμένες τιμές. Πιο συγκεκριμένα, οι κατασκευαστές που προσφέρονται είναι οι παρακάτω:

- public OverlappingClustering(Graph<V, E> graph, ThreadPoolExecutor executor, int radius)
  - Η πλήρης μορφή του κατασκευαστή που δέχεται τιμές για όλα τα πεδία της κλάσης.

- public OverlappingClustering(Graph<V, E> graph, int parallelism, int radius)
  - Η μορφή του κατασκευαστή που δημιουργεί μόνη της τον executor νημάτων, με την χρήση όμως της τιμής παραλληλοποίησης που δίνεται.

- public OverlappingClustering(Graph<V, E> graph)
  - Η πιο απλή μορφή του κατασκευαστή που χρησιμοποιεί τις προκαθορισμένες τιμές για radius και καλή την προηγούμενη με την προκαθορισμένη τιμή παραλληλοποίησης.

Τέλος, ας περιγράψουμε την βασική λειτουργία της κλάσης, που ως σκοπό έχει την εξαγωγή των κοινοτήτων του γράφου από τα στοιχεία που συλλέγονται μέσω των egonet και τυπικά υλοποιείται στην μέθοδο getClustering. Αποφασίστηκε για την καλύτερη κατανόηση του κώδικα η εκτέλεση της συγκεκριμένης μεθόδου να διαιρεθεί σε τρία τμήματα - μεθόδους (findFriendshipGroups, removeProperSubsets, mergeCloseSets) από τα οποία το κάθε ένα εκτελεί μια ξεχωριστή λειτουργία. Επιπλέον, οι πρώτες δύο μέθοδοι εκτελούνται παράλληλα σε νήματα, αν η συσκευή εκτέλεσης το επιτρέπει, αλλά στην μέθοδο παραλληλοποίησης θα γίνει αναφορά αργότερα παρακάτω. Ο ορισμός των μεθόδων σε επίπεδο Java είναι ο παρακάτω:

- void findFriendshipGroups()
  - Η μέθοδος που χρησιμοποιείται για την εύρεση των φιλικών groups και ουσιαστικά γεμίζει την λίστα-πεδίο friendshipGroups της κλάσης.

- void removeProperSubsets()
  - Η μέθοδος που χρησιμοποιείται για την αφαίρεση από τα φιλικά groups των γνήσιων υποσυνόλων και ουσιαστικά γεμίζει την λίστα-πεδίο superGroups της κλάσης.

- void mergeCloseSets()
  - Η μέθοδος που χρησιμοποιείται για να πραγματοποιηθούν τα εναλλασσόμενες συγχωνεύσεις ώστε να προκύψουν οι κοινότητες και ουσιαστικά γεμίζει την λίστα-πεδίο mergedGroups της κλάσης.

3.3.1 Περιγραφή Τμήματος Εύρεσης Φιλικών Ομάδων

Η πρώτη κλάση έχει ως στόχο την εύρεση των φιλικών group όπως έχει αναλυθεί στην περιγραφή του αλγορίθμου εύρεσης επικαλυπτόμενων κοινοτήτων. Για μια κορυφή του γράφου που ορίζεται προσωρινά ως ego, ξεκινάμε δημιουργώντας έναν κενό γράφο target χρησιμοποιόντας τα εργαλεία που παρέχει η βιβλιοθήκη JGraphT. Στην συνέχεια, δημιουργείται ένα αντικείμενο της γεννήτριας EgonetGraphGenerator όπως καθορίστηκε προηγουμένως, δίνοντας false τιμή στο πεδίο includeEgo καθώς για τον υπολογισμό των φιλικών ομάδων δεν μας ενδιαφέρει να κρατήσουμε στο egonet την κεντρική κορυφή. Στην συνέχεια εκτελείται η γεννήτρια και βρίσκει το επιθυμητό egonet, από το οποίο πλέον πρέπει να εξάγουμε τα φιλικά group μέσω της εύρεσης των συνδεδεμένων συνιστωσών. Για την εύρεση των συνδεδεμένων τμημάτων, η βιβλιοθήκη JGraphT παρέχει μια κλάση που καλείται ConnectivityInspector, ενώ μαζί με τον κατασκευαστή αντικειμένων της ορίζεται ως εξής:

- public class ConnectivityInspector<V, E>
  implements GraphListener<V, E>
  - V – Ο τύπος κορυφών του γράφου.
  - E – Ο τύπος ακμών του γράφου.

- public ConnectivityInspector(Graph<V, E> g)
  - Κατασκευαστής της κλάσης.

- public List<Set<V>> connectedSets()
  - Μέθοδος που επιστρέφει τα συνδεδεμένα μέρη.

Κατά την δημιουργία ενός αντικειμένου της παραπάνω κλάσης βρίσκονται άμεσα και οι συνδεδεμένες συνιστώσες που αναζητούσαμε οι οποίες είναι προσβάσιμες μέσω κλήσης της μεθόδου connectedSets. Ως τελευταίο βήμα, στις συνδεδεμένες συνιστώσες (σύνολα) που επιστρέφονται μέσω αυτής της διαδικασίας προσθέτουμε στο τέλος και την κορυφή ego και πλέον έχουμε τις φιλικές ομάδες της κορυφής ego, οι οποίες προστίθενται στην λίστα

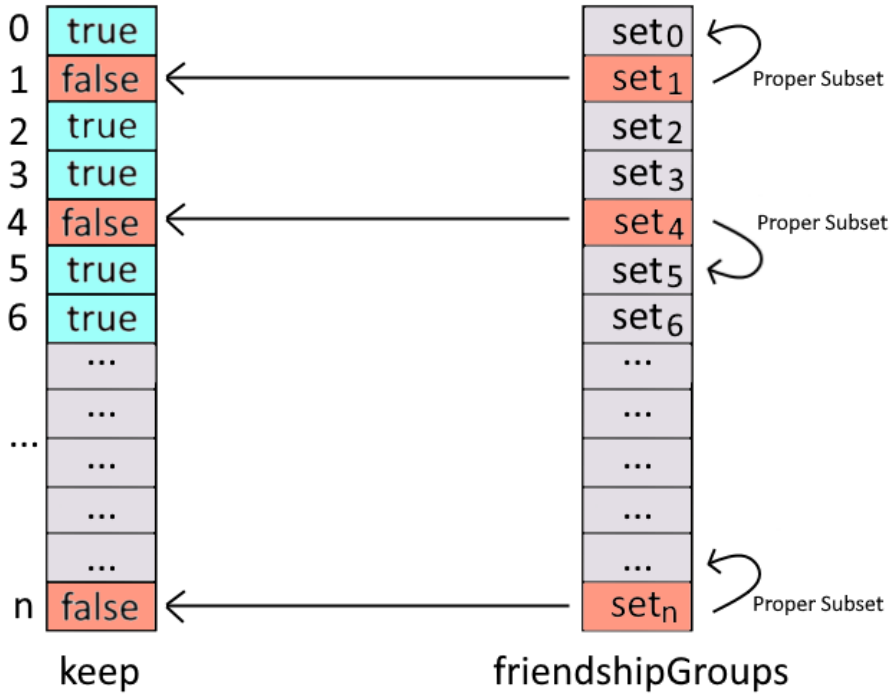
friendshipGroups της κλάσης. Αξίζει να τονίσουμε, ότι στο στάδιο ανάπτυξης, για να αποφευχθούν περαιτέρω βήματα για την αφαίρεση πανομοιότυπων συνόλων στην επόμενη μέθοδο, αποφασίστηκε εσωτερικά της μεθόδου οι φιλικές ομάδες αρχικά να τοποθετούνται σε ένα σύνολο συνόλων τύπου Set<Set<V>> groups, το οποίο λόγω ιδιοτήτων και υλοποίησης της Java δεν επιτρέπει πανομοιότυπα αντικείμενα. Η διαδικασία επαναλαμβάνεται για κάθε κορυφή του γράφου ώστε να συλλέξουμε όλα τα φιλικά group που μπορούν να προκύψουν.

3.3.2 Περιγραφή Τμήματος Απαλοιφής Γνήσιων Υποσυνόλων

Το δεύτερο τμήμα έχει ως στόχο την απαλοιφή των πανομοιότυπων συνόλων αλλά και των γνήσιων υποσυνόλων από την λίστα friendshipGroups που βρήκαμε στο προηγούμενο βήμα. Όπως αναφέρθηκε στην προηγούμενη παράγραφο, κατά την εύρεση των φιλικών ομάδων λόγω σχεδιασμού δεν επιτρέπονται πανομοιότυπα σύνολα, οπότε μας ενδιαφέρει μόνο η αφαίρεση των γνήσιων υποσυνόλων. Για να αποφανθούμε αν ένα σύνολο είναι γνήσιο υποσύνολο κάποιου άλλου, χρησιμοποιούμε την μέθοδο isProperSubset, που ελέγχει την συνθήκη για τα γνήσια υποσύνολα η οποία για να επιστρέψει αλήθεια για δύο σύνολα S_1 και S_2 απαιτεί να ισχύει ότι το S_2 περιέχει όλο το S_1 αλλά ταυτόχρονα δεν ταυτίζεται με αυτό. Η μέθοδος είναι λογικού τύπου και επιστρέφει αληθές ή ψευδές συμπέρασμα. Τυπικά, η μορφή της μεθόδου σε Java είναι η παρακάτω:

- boolean isProperSubset(Set<V> setA, Set<V> setB)
  - setA το πρώτο σύνολο προς σύγκριση.
  - setB το δεύτερο σύνολο προς σύγκριση.

Ας επιστρέψουμε όμως στην λειτουργία του συγκεκριμένου τμήματος. Για ένα σύνολο της λίστας συνόλων που υπολογίσαμε στο πρώτο βήμα (friendshipGroups), διατρέχουμε όλα τα υπόλοιπα σύνολα της λίστας και με χρήση της μεθόδου που προαναφέραμε ελέγχεται αν ισχύει η συνθήκη γνήσιου υποσυνόλου μεταξύ του επιλεγμένου συνόλου και οποιουδήποτε άλλου συνόλου στην λίστα. Διατηρούμε έναν πίνακα λογικών τιμών keep μεγέθους ίσο με τον αριθμό των συνόλων που βρίσκουμε στην λίστα friendshipGroups, ο οποίος αρχικοποιείται με αλήθεια σε κάθε θέση του, και κατά την εκτέλεση ισχύει ότι: αν ένα σύνολο S_n είναι γνήσιο υποσύνολο ενός συνόλου S_k τότε η τιμή του πίνακα keep στην θέση n πρέπει να γίνει ψευδής.



Εικόνα 16: Απεικόνιση Πίνακα Διατήρησης

Ο πίνακας αυτός μας επιτρέπει να γνωρίζουμε στο τέλος της μεθόδου πια σύνολα πρέπει να απαλείφουν από την λίστα συνόλων. Στην περίπτωση που βρεθεί ένα τέτοιο σύνολο, τότε ορίζουμε στον λογικό πίνακα την θέση που το αντιπροσωπεύει τιμή ψεύδους, και σταματάμε την προσπέλαση της λίστας συνόλων για αυτό. Επαναλαμβάνουμε την παραπάνω διαδικασία για κάθε σύνολο της λίστας συνόλων και στο τέλος επιτυγχάνεται η απαλοιφή των γνήσιων υποσυνόλων που επιθυμούσαμε.

3.3.3 Περιγραφή Τμήματος Ένωσης Κοντινών Συνόλων

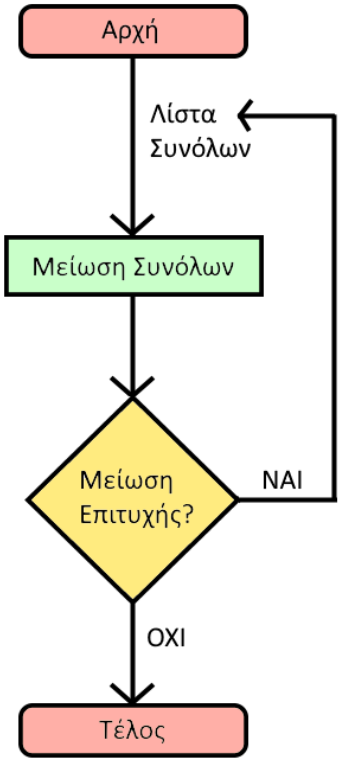
Η τρίτη κλάση του αλγορίθμου ασχολείται με την ένωση των συνόλων που προέκυψαν από τις δύο προηγούμενες, με βάση το πόσο «κοντά» μπορεί να θεωρηθεί ότι είναι δύο σύνολα. Απώτερος σκοπός του βήματος είναι η αποτελεσματική τελική μείωση των συνόλων ώστε να παραμείνουν μόνο τα ελάχιστα δυνατά, που θα αποτελούν τις επικαλυπτόμενες κοινότητες που αναζητούμε. Γενικότερα, στην διάρκεια της ανάπτυξης της υλοποίησης για την συγκεκριμένη πτυχιακή εργασία αποφασίστηκε να χρησιμοποιηθεί μια σχετικά «άπληστη» προσέγγιση χωρίς παραλληλοποίηση. Για να αποφανθούμε για το αν δύο σύνολα βρίσκονται «κοντά» μεταξύ τους, χρησιμοποιούμε την μέθοδο areCloseBy, που ελέγχει αν για δύο σύνολα S_1 και S_2 όπου $S_1 >= S_2$ ισχύει $|S_1 \cup S_2| = |S_2| - 1$. Αξίζει να σημειωθεί ότι για τους σκοπούς της υλοποίησης, η μέθοδος

φροντίζει επίσης να ισχύει ότι το πρώτο σύνολο είναι μικρότερο από το δεύτερο ενώ σε αντίθετη περίπτωση τα αντιμεταθέτει. Στην συνέχεια υπολογίζει την ένωση τους για να μπορεί να ελέγξει την συνθήκη που προαναφέραμε. Η μέθοδος είναι λογικού τύπου και επιστρέφει αληθές ή ψευδές ανάλογα με το αν τα δύο σύνολα είναι «κοντά» μεταξύ τους. Η μορφή της μεθόδου σε Java είναι η παρακάτω:

- boolean areCloseBy(Set<V> setA, Set<V> setB)
  - setA το πρώτο σύνολο προς σύγκριση.
  - setB το δεύτερο σύνολο προς σύγκριση.

Ας επικεντρωθούμε και πάλι στην λειτουργία της μεθόδου που πραγματοποιεί τις ενώσεις των «κοντινών» συνόλων. Όπως προαναφέρθηκε, η προσέγγιση είναι αρκετά «άπληστη» αλλά παρόλα αυτά αποδίδει τα επιθυμητά αποτελέσματα. Θα περιγράψουμε την διαδικασία ενώσεων για μία λίστα συνόλων που στην συνέχεια αν απαιτείται επαναλαμβάνεται στην νέα λίστα συνόλων που θα προκύψει. Όπως και στο προηγούμενο τμήμα της απαλοιφής των γνήσιων υποσυνόλων, υπάρχει η ανάγκη χρήσης ενός λογικού πίνακα που καλείται keep μεγέθους ίσο με τον αριθμό των συνόλων στην λίστα μας, ο οποίος αρχικοποιείται με αλήθεια σε κάθε θέση του και ισχύει ότι η τιμή μετατρέπεται σε ψευδής όταν το σύνολο δεν πρέπει πια να διατηρηθεί. Για κάθε σύνολο της λίστας, διατρέχουμε όλα τα υπόλοιπα που ανήκουν σε αυτήν, και συγκρίνουμε με την μέθοδο areCloseBy την περίπτωση «κοντινού» συνόλου. Σε περίπτωση που βρεθεί ένα τέτοιο σύνολο που θα ενωθεί με το πρώτο, πραγματοποιείται ένωση και θέτεται στον πίνακα keep ψευδής τιμή στην θέση του δεύτερου συνόλου και ως αποτέλεσμα πραγματοποιείται η ένωση και στην επόμενη εκτέλεση, το δεύτερο σύνολο δεν θα διατηρηθεί. Η διαδικασία επαναλαμβάνεται για κάθε σύνολο της λίστας ώστε να γίνουν οι πιθανές ενώσεις, και στο τέλος τροποποιείται η λίστα, με την βοήθεια της πληροφορίας που παρέχει ο πίνακας keep, ώστε να αφαιρεθούν τα σύνολα που έχουν ενσωματωθεί εντός άλλων.

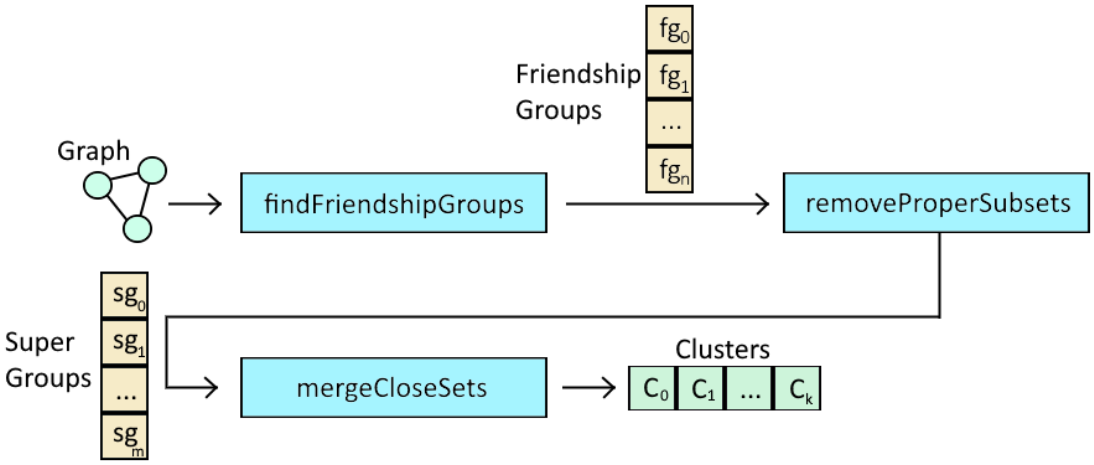
Ως έξοδο από την παραπάνω διαδικασία, έχουμε μια νέα λίστα συνόλων που πιθανώς μπορεί να ξανά περάσει από την ίδια διαδικασία ώστε να προκύψουν νέες ενώσεις. Για να πετύχουμε την μέγιστη μείωση των συνόλων, ορίζουμε ότι η διαδικασία πραγματοποιείται ξανά, μέχρις ότου να μην υπάρχουν πλέον πιθανές ενώσεις. Η λογική του συγκεκριμένου τμήματος του αλγορίθμου που πραγματοποιεί τις ενώσεις μπορεί να γίνει ευκολότερα αντιληπτή από το παρακάτω απλοποιημένο διάγραμμα ροής.



Εικόνα 17: Διάγραμμα Ροής Τμήματος Ενώσεων

Συνοψίζοντας την λειτουργία των τριών τμημάτων της υλοποίησης του αλγορίθμου παρατηρούμε τρία βασικά στάδια εκτέλεσης. Αρχικά παράγονται με την βοήθεια των ego networks οι φιλικές ομάδες για κάθε κορυφή και αποθηκεύονται σε μια λίστα που δεν επιτρέπει πανομοιότυπες εγγραφές, με τη μορφή συνόλων. Στην συνέχεια, από την προηγούμενη λίστα, αφαιρούνται τα σύνολα που είναι γνήσια υποσύνολα άλλων συνόλων στην λίστα. Τέλος, πραγματοποιούνται επαναλαμβανόμενες ενώσεις των συνόλων με κριτήριο το πόσο κοντά είναι ένα σύνολο με τα υπόλοιπα της λίστας έως ότου να μην υπάρχουν άλλες πιθανές ενώσεις που μπορούν να πραγματοποιηθούν.

Στην παρακάτω εικόνα, μπορούμε να κατανοήσουμε σχηματικά την λειτουργία του αλγορίθμου από το πρώτο βήμα της εισαγωγής του γράφου έως την τελική αποκάλυψη των κοινοτήτων του.



Εικόνα 18: Στάδια Εκτέλεσης Αλγορίθμου

3.4 Προσθήκη Δυνατότητας Παράλληλης Εκτέλεσης

Σε αυτό το σημείο θα γίνει μια αναφορά στην μέθοδο παραλληλοποίησης που εφαρμόστηκε στην υλοποίηση του αλγορίθμου σε δεύτερη φάση με στόχο την βελτίωση της απόδοσης του ειδικά σε μεγαλύτερους και πυκνότερους γράφους. Πραγματοποιήθηκαν προσπάθειες παραλληλοποίησης και για τα τρία τμήματα που προαναφέρθηκαν αλλά στην τελική μορφή της υλοποίησης περιλαμβάνεται δυνατότητα παραλληλοποίησης μόνο για τα πρώτα δύο τμήματα, μιας και η παράλληλη λογική μπορεί να αλλοιώσει τα αποτελέσματα του τρίτου «άπληστου» τμήματος των ενώσεων συνόλων. Θα αναφερθούμε και πάλι ξεχωριστά για κάθε τμήμα του αλγορίθμου για λόγους σαφήνειας.

Αρχικά αξίζει να γίνει μια αναφορά στην κλάση-υπηρεσία που χρησιμοποιήθηκε στην υλοποίηση του αλγορίθμου για να επιτευχθεί η σωστή διαχείριση των νημάτων. Η κλάση, μέρος της Java, καλείται ExecutorCompletionService, ενώ υλοποιεί την διεπαφή CompletionService που είναι μια γενική προσέγγιση υπηρεσίας διαχείρισης διαδικασιών που εκτελούνται παράλληλα σε νήματα και η μορφή της είναι η παρακάτω:

- public class ExecutorCompletionService<V>
  - V – Ο τύπος διαδικασιών (tasks) που θα χρησιμοποιηθεί την κάθε φορά.

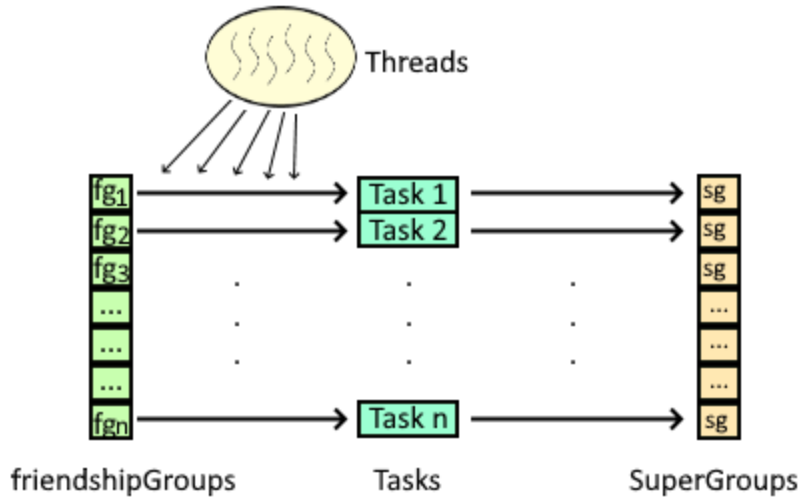
Οι υπολογισμοί των πρώτων δύο τμημάτων της εύρεσης των φιλικών ομάδων και της απαλοιφής των γνήσιων υποσυνόλων είναι πλήρως ανεξάρτητοι μεταξύ τους. Έτσι μας επιτρέπεται η χρήση της παραπάνω κλάσης, μιας και ο τρόπος διαχείρισης των επιστροφών διαδικασιών έχει την λογική διαχείρισης του νήματος που τελειώνει πρώτο την εργασία του σε κάθε δεδομένη στιγμή, ανεξαρτήτου σειράς εισαγωγής στην υπηρεσία. Οι διαδικασίες προστίθενται στην υπηρεσία, οι οποίες στην συνέχεια αναθέτονται σε νήματα για την εκτέλεση της εργασίας που τους αναλογεί. Ο αριθμός των μέγιστων νημάτων που μπορούν να εκτελεστούν ταυτοχρόνως δίνεται από την παράμετρο parallelism του κατασκευαστή που έχει προαναφερθεί. Η υπηρεσία χρησιμοποιεί μια ουρά στην οποία μπορούν να προστεθούν όλες οι διαδικασίες που θέλουμε να εκτελέσουμε συνολικά και φροντίζει να τις εκτελέσει όλες αναλόγως με τον αριθμό νημάτων που τις έχουν διατεθεί.

Στο πρώτο τμήμα της υλοποίησης χρησιμοποιήθηκε η παραπάνω υπηρεσία και υλοποιήθηκε η κλάση διαδικασία FriendshipGroupsTask η οποία βρίσκει το egonet και τις φιλικές ομάδες μιας κορυφής την φορά. Η κλάση-τύπος διαδικασιών που χρησιμοποιήθηκε είναι η Callable, ώστε να μπορούν να επιστρέφονται στο κύριο πρόγραμμα οι φιλικές ομάδες που θα υπολογιστούν, ενώ ο τύπος επιστροφής είναι ένα σύνολο συνόλων που αναπαριστά τις φιλικές ομάδες της κορυφής. Η μορφή της κλάσης Callable και της διαδικασίας είναι η ακόλουθη:

- public interface Callable<V>
  - V – Ο τύπος της μεταβλητής που θα επιστραφεί.

- class FriendshipGroupsTask implements Callable<Set<Set<V>>>
  - V – Ο τύπος κορυφών του γράφου και συνεπώς και των τιμών που περιέχονται στις φιλικές ομάδες.

Η εκτέλεση των υπολογισμών είναι παρόμοια με αυτή που προαναφέρθηκε στην περιγραφή της υλοποίησης με την ενσωμάτωση της υπηρεσίας και των διαδικασιών παραλληλοποίησης. Για κάθε κορυφή του γράφου προστίθεται στην υπηρεσία μια διαδικασία της παραπάνω μορφής. Στην συνέχεια, αφού ολοκληρωθεί η προσθήκη όλων των διαδικασιών στην ουρά, η υπηρεσία τις εκτελεί σε νήματα και όταν ένα από τα αυτά τελειώσει την εργασία του, η μέθοδος συλλέγει τα δεδομένα που επιστρέφονται και τα προσθέτει στην συνολική λίστα των φιλικών ομάδων όπως προαναφέρθηκε, χωρίς να επηρεάζεται το τελικό αποτέλεσμα. Στην παρακάτω εικόνα παρουσιάζεται γραφικά η λειτουργία του συγκεκριμένου τμήματος.

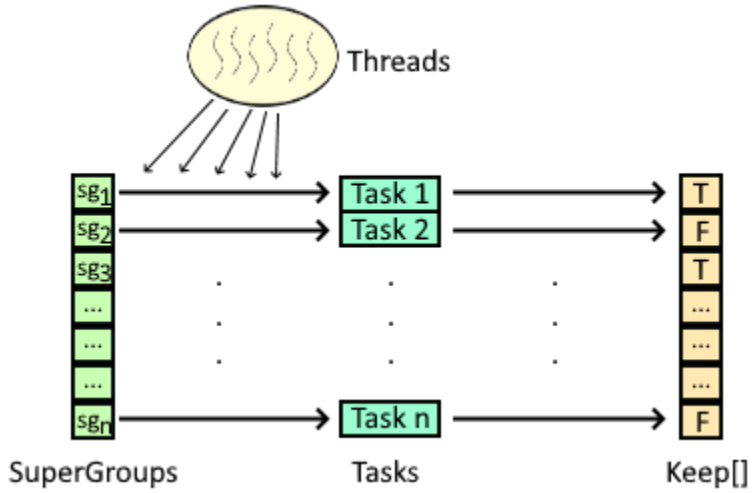


Εικόνα 19: Χρήση Νημάτων στο Πρώτο Τμήμα του Αλγορίθμου

Στο δεύτερο τμήμα της υλοποίησης χρησιμοποιήθηκε και πάλι η προηγούμενη υπηρεσία με ένα διαφορετικό τύπο διαδικασιών που ονομάστηκε ProperSubsetsTask η οποία δέχεται ένα σύνολο της λίστας των φιλικών ομάδων και μπορεί να αποφανθεί αν πρέπει να διατηρηθεί ή αν αποτελεί γνήσιο υποσύνολο κάποιοι άλλου συνόλου, περίπτωση στην οποία απαλείφεται. Ο τύπος επιστροφής αυτή την φορά είναι μια Optional τιμή ακεραίου. Πιο συγκεκριμένα για την κλάση Optional, μια μεταβλητή του τύπου της μπορεί να περιέχει μια μεταβλητή στο εσωτερικό της ή και όχι, η οποία μπορεί να προσπελαστεί με συγκεκριμένες μεθόδους. Η μορφή της κλάσης Optional και της διαδικασίας είναι η ακόλουθη:

- public final class Optional<T>
  - boolean isPresent()
    - Μέθοδος που επιστρέφει αλήθεια αν υπάρχει τιμή στην Optional μεταβλητή.
  - T get()
    - Μέθοδος που επιστρέφει την τιμή που περιέχει η Optional μεταβλητή.
- Class ProperSubsetsTask implements Callable<Integer>
  - Η κλάση επιστρέφει Optional Integer ο οποίος αν υπάρχει αντιπροσωπεύει την θέση του συνόλου στον πίνακα διατήρησης.

Όπως και προηγουμένως, η εκτέλεση των υπολογισμών είναι παρόμοια με αυτή που αναφέρθηκε στην περιγραφή της υλοποίησης με την ενσωμάτωση της υπηρεσίας και των διαδικασιών παραλληλοποίησης. Για κάθε σύνολο των φιλικών ομάδων, προστίθεται στην υπηρεσία μια διαδικασία της παραπάνω μορφής. Στην συνέχεια, αφού ολοκληρωθεί η προσθήκη όλων των διαδικασιών στην ουρά, η υπηρεσία τις εκτελεί σε νήματα και όταν ένα από αυτά τελειώσει την εργασία του, αν η Optional μεταβλητή περιέχει τιμή-θέση συνόλου, η μέθοδος ενημερώνει τον πίνακα διατήρησης keep στην συγκεκριμένη θέση. Οι υπολογισμοί είναι και πάλι ανεξάρτητοι οπότε το αποτέλεσμα δεν επηρεάζεται από την τυχαία σειρά επιστροφής των νημάτων. Στην παρακάτω εικόνα παρουσιάζεται γραφικά η λειτουργία του συγκεκριμένου τμήματος.



Εικόνα 20: Χρήση Νημάτων στο Δεύτερο Τμήμα του Αλγορίθμου

Στο τρίτο τμήμα τις υλοποίησης, αναπτύχθηκε ένας τρόπος προσθήκης δυνατότητας παραλληλοποίησης για την μέθοδο mergeCloseSets, ο οποίος όμως δεν διατηρήθηκε στην τελική υλοποίηση. Πιο συγκεκριμένα, έγινε μια προσπάθεια συλλογής των ενώσεων που πρέπει να γίνουν ανάλογα με τα σύνολα που είναι "κοντά" μεταξύ τους. Τα ζεύγη ενώσεων που προέκυψαν αρκετές φορές δεν ήταν ανεξάρτητα μεταξύ τους και δεν επέτρεπαν παράλληλη εκτέλεση. Έτσι έπρεπε τα ζεύγη που προέκυψαν να διαχωριστούν σε ανεξάρτητες ενώσεις πριν πραγματοποιηθεί οποιαδήποτε ένωση παράλληλα. Τα αποτελέσματα παρουσίαζαν αλλοιώσεις σε σύγκριση με την μη παράλληλη έκδοση της μεθόδου, μιας και κάποιες αρχικές ενώσεις μπορεί να μην εκτελεστούν ποτέ, αποδίδοντας μεγαλύτερο αριθμό κοινοτήτων. Επιπλέον, δεν υπήρξε ουσιαστική βελτίωση στον χρόνο εκτέλεσης μιας και έπρεπε να γίνουν επιπλέον βήματα για τον υπολογισμό των ενώσεων αλλά και για την εύρεση των ανεξάρτητων αυτών. Συνεπώς, αποφασίστηκε να μην συμπεριληφθεί στην τελική υλοποίηση του αλγορίθμου, μιας και μια ορθότερη και αποδοτικότερη υλοποίηση του συγκεκριμένου τμήματος παράλληλα θα απαιτούσε και την ριζική αλλαγή του.

Κεφάλαιο 4ο Πειραματισμός

4.1 Εισαγωγή

Στο συγκεκριμένο κεφάλαιο θα αναφερθούμε αρχικά στις θετικές επιπτώσεις της χρήσης νημάτων στην υλοποίηση του αλγορίθμου μέσω μερικών τυπικών πειραμάτων που πραγματοποιήθηκαν ώστε να βρεθούν προσεγγιστικά οι χρόνοι εκτέλεσης με χρήση νημάτων ή χωρίς. Επιπλέον θα παρουσιάσουμε αναλυτικά τα αποτελέσματα της εφαρμογής του αλγορίθμου και συνεπώς της εύρεσης κοινοτήτων σε μερικούς γράφους. Για της ανάγκες οπτικοποίησης των γράφων αλλά και των κοινοτήτων που βρίσκονται από τον αλγόριθμο σε αυτούς χρησιμοποιήθηκε το πακέτο Graphviz. Το Graphviz είναι ένα πακέτο ανοιχτού λογισμικού, κύρια χρήση του οποίου είναι η γραφική απεικόνιση γράφων που πρώτα πρέπει να περιγράφουν στην γλώσσα περιγραφής γράφων DOT. Το πακέτο παρέχει διάφορους αλγόριθμους απεικόνισης γράφων τους οποίος ο χρήστης μπορεί να επιλέξει και να χρησιμοποιήσει.

4.2 Παραδείγματα Χρόνων Εκτέλεσης

Για να υπολογιστούν τυπικά οι μέσοι χρόνοι εκτέλεσης του προγράμματος, είτε με την χρήση νημάτων είτε όχι, αποφασίστηκε να εκτελεστεί ο αλγόριθμος σε δύο γράφους διαφορετικού μεγέθους. Ο πρώτος γράφος που χρησιμοποιήθηκε απαρτίζεται από είκοσι κορυφές, ενώ ο δεύτερος από εκατό πενήντα κορυφές. Οι δοκιμές έγιναν στην περίπτωση μη χρήσης νημάτων, στην περίπτωση χρήσης τεσσάρων νημάτων και τέλος στην περίπτωση χρήσης οκτώ νημάτων. Ο στόχος της συγκεκριμένης επιλογής γράφων και αριθμού νημάτων είναι να αναδείξει τα επαυξημένα οφέλη της χρήσης όσο το δυνατών περισσότερων νημάτων ειδικότερα σε μεγαλύτερους γράφους. Πραγματοποιήθηκαν λοιπόν πειράματα με τα παραπάνω δεδομένα σε 10 επαναλήψεις για κάθε μία από τις περιπτώσεις που προέκυψαν.

Στους παρακάτω πίνακες παρατηρούμε αναλυτικά τους χρόνους εκτέλεσης, καθώς και τον μέσο όρο αυτών με είσοδο τον πρώτο γράφο των 20 κορυφών. Δεδομένου ότι ο γράφος εισόδου είναι αρκετά μικρός, αναμένεται η βελτίωση του χρόνου εκτέλεσης με την χρήση νημάτων να μην είναι ιδιαίτερα σημαντική. Συνεπώς, οι μέσοι χρόνοι εκτέλεσης που καταγράφηκαν για κάθε μια περίπτωση του πρώτου γράφου παρουσιάζουν μόνο μια μικρή μεταβολή της τάξης των 3.5 ms.

X	1	2	3	4	5	6	7	8	9	10
Single Core	44 ms	49 ms	47 ms	46 ms	56 ms	46 ms	44 ms	46 ms	48 ms	42 ms
4 Threads	38 ms	41 ms	44 ms	44 ms	42 ms	48 ms	44 ms	45 ms	42 ms	47 ms
8 Threads	37 ms	44 ms	46 ms	42 ms	43 ms	42 ms	44 ms	45 ms	43 ms	47 ms

Πίνακας 3: Χρόνοι Εκτέλεσης Πρώτου Παραδείγματος

Threads	Time
Single Core	46.8 ms
4 Threads	43.5 ms
8 Threads	43.3 ms

Πίνακας 4: Μέσοι Χρόνοι Εκτέλεσης Δεύτερου Παραδείγματος

Στους παρακάτω πίνακες παρατηρούμε αναλυτικά τους χρόνους εκτέλεσης, καθώς και τον μέσο όρο αυτών με είσοδο τον δεύτερο γράφο των 150 κορυφών. Δεδομένου ότι ο γράφος εισόδου είναι μεγαλύτερου μεγέθους, αναμένεται η βελτίωση του χρόνου εκτέλεσης με την χρήση νημάτων να είναι περισσότερο αισθητή. Συνεπώς, οι μέσοι χρόνοι εκτέλεσης που καταγράφηκαν για κάθε μια περίπτωση του δεύτερου γράφου παρουσιάζουν μεγαλύτερες μεταβολές, της τάξης των 15.3 ms μεταξύ της μη χρήσης νημάτων και της χρήσης τεσσάρων, ενώ ακόμα μεγαλύτερη αλλαγή παρατηρείται μεταξύ της πρώτης περίπτωσης και αυτής της χρήσης οκτώ νημάτων, δηλαδή της τάξης των 20 ms.

X	1	2	3	4	5	6	7	8	9	10
Single Core	114 ms	112 ms	108 ms	110 ms	111 ms	105 ms	105 ms	115 ms	112 ms	113 ms
4 Threads	98 ms	90 ms	94 ms	95 ms	85 ms	95 ms	95 ms	99 ms	100 ms	101 ms
8 Threads	94 ms	86 ms	90 ms	90 ms	93 ms	94 ms	88 ms	95 ms	93 ms	82 ms

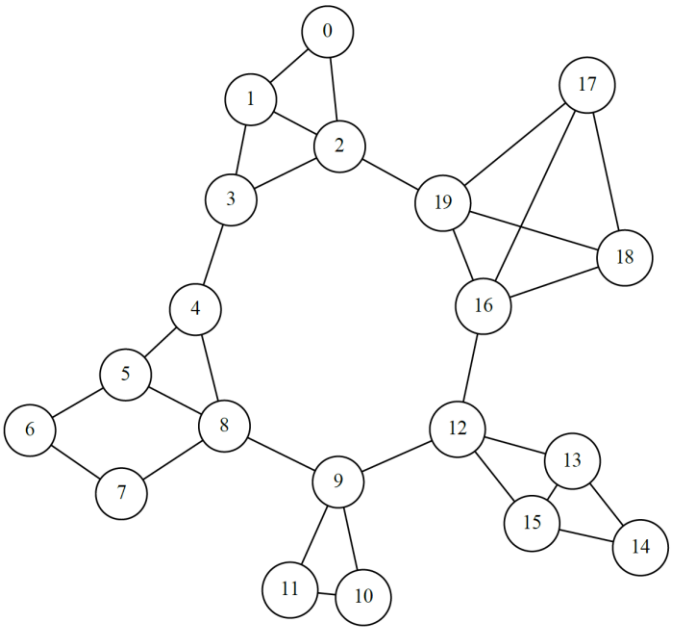
Πίνακας 5: Χρόνοι Εκτέλεσης Δεύτερου Παραδείγματος

Threads	Time
Single Core	110.5 ms
4 Threads	95.2 ms
8 Threads	90.5 ms

Πίνακας 6: Μέσοι χρόνοι εκτέλεσης δεύτερου γράφου

4.3 Παραδείγματα Εύρεσης Κοινοτήτων

Σε αυτό το σημείο θα παρουσιάσουμε ένα πλήρες παράδειγμα αναλυτικά σε βήματα. Εισάγουμε στον αλγόριθμο έναν απλό γράφο είκοσι κορυφών ο οποίος παρουσιάζεται με την βοήθεια του Graphviz στην παρακάτω εικόνα:



Εικόνα 21: Απεικόνιση Γράφου Πρώτου Παραδείγματος

Εκτελούμε τον αλγόριθμο σε ξεχωριστά τμήματα για λόγους σαφήνειας του παραδείγματος. Αρχικά, εκτελούμε το πρώτο κομμάτι της εύρεσης φιλικών ομάδων για κάθε κορυφή. Η συνολική λίστα φιλικών ομάδων για τις κορυφές του παραδείγματος είναι η ακόλουθη:

0,1,2	0,1,2,3	1,2,3	16,17,18,19	3,4	2,19	7,8	5,6	6,7
13,14,15	4,5,8	8,9	12,13,15	9,12	12,13,14,15	16,12	9,10,11	

Πίνακας 7: Λίστα Φιλικών Ομάδων Πρώτου Παραδείγματος

Αναμενόμενα, λόγω των επιλογών της υλοποίησης δεν παρουσιάζονται πανομοιότυπα σύνολα στην παραπάνω λίστα φιλικών ομάδων, όμως προφανώς υπάρχουν σύνολα τα οποία αποτελούν γνήσια υποσύνολα άλλων συνόλων. Ο αριθμός των φιλικών ομάδων είναι δεκαεπτά. Σε αυτό το σημείο προχωράμε στην εκτέλεση του δεύτερου τμήματος, δηλαδή της απαλοιφής γνήσιων υποσυνόλων από την λίστα φιλικών ομάδων. Η συνολική λίστα υπερσυνόλων που προκύπτει μετά την εκτέλεση είναι η ακόλουθη:

0,1,2,3	16,17,18,19	3,4	5,6	6,7	16,12
7,8	4,5,8	8,9	2,19	12,13,14,15	9,10,11

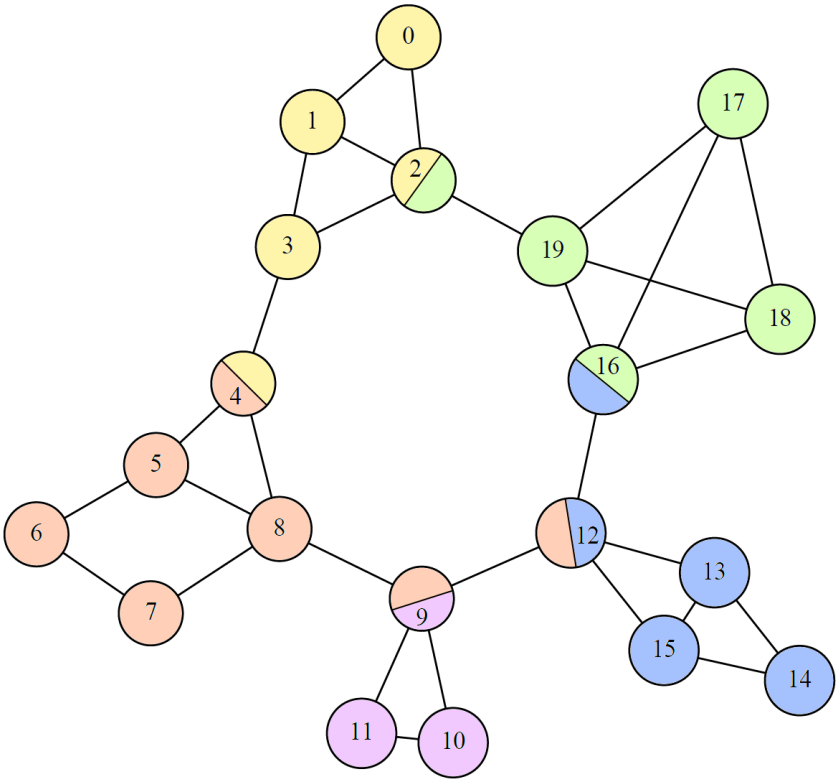
Πίνακας 8: Υπερσύνολα Πρώτου Παραδείγματος

Όπως περιμέναμε, προκύπτουν σύνολα τα οποία είναι μοναδικά μεταξύ τους χωρίς ζεύγη γνήσιων υποσυνόλων. Ο αριθμός των υπερσυνόλων είναι δεκατρία. Στην συνέχεια θα προχωρήσουμε στην εκτέλεση του τελευταίου τμήματος του αλγορίθμου που ενώνει τα σύνολα που βρίσκονται «κοντά» μεταξύ τους, σε ξεχωριστά βήματα για την κάθε επανάληψη ενώσεων.

0,1,2,3,4	16,17,18,2,19	4,5,6,7,8,9,12	16,12,13,14,15	9,10,11

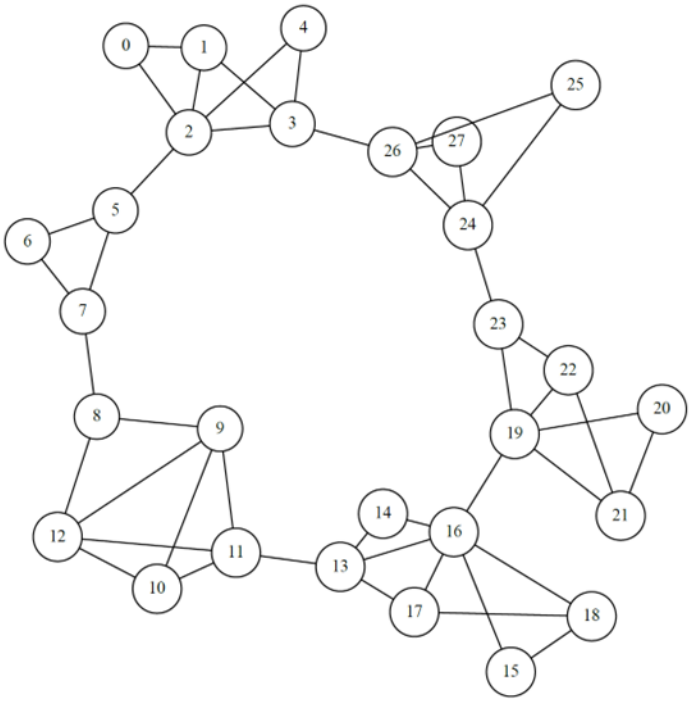
Πίνακας 9: Λίστα Κοινοτήτων Πρώτου Παραδείγματος

Παρατηρούμε ότι προκύπτουν πέντε κοινότητες στον γράφο που χρησιμοποιήθηκε, με εμφανή σημεία επικάλυψης των κοινοτήτων. Συγκεκριμένα, παρατηρείται επικάλυψη στις κορυφές 2, 4, 9, 12 και 16. Ο γράφος χωρισμένος σε κοινότητες όπως προέκυψαν από την εκτέλεση του αλγορίθμου παρουσιάζεται στην παρακάτω εικόνα:



Εικόνα 22: Απεικόνιση Κοινοτήτων Πρώτου Παραδείγματος

Τέλος, θα παρουσιάσουμε ένα ακόμα παράδειγμα εκτέλεσης του αλγορίθμου με έναν λίγο μεγαλύτερο γράφο ως είσοδο, μεγέθους 28 κορυφών με λιγότερες λεπτομέρειες. Ο γράφος παρουσιάζεται με την βοήθεια του Graphviz στην παρακάτω εικόνα:



Εικόνα 23: Απεικόνιση Γράφου Δεύτερου Παραδείγματος

Στο πρώτο μέρος της εκτέλεσης του αλγορίθμου προκύπτουν οι παρακάτω φιλικές ομάδες για το σύνολο των κορυφών του παραπάνω γράφου:

16,17,18,13	19,22,23	16,17,18,15	0,1,2	0,1,2,3	2,5	2,3,4
1,2,3,4	0,1,2,3,4	24,25,26	24,26,27	7,8	5,6,7	19,20,21,22
19,21,22,23	11,13	3,26	8,9,12	16,17,1,13,14,15	16,19	24,25,26,27
19,20,21,22,23	16,13,14	23,24	16,18,15	8,9,10,11,12	16,17,13,14	19,20,21
24,25,26						

Πίνακας 10: Λίστα Φιλικών Ομάδων Δεύτερου Παραδείγματος

Όπως και προηγουμένως, προκύπτουν σύνολα χωρίς να υπάρχουν πανομοιότυπα μεταξύ τους, ενώ παρατηρούνται σύνολα τα οποία είναι γνήσια υποσύνολα άλλων συνόλων. Έτσι προχωράμε στην εκτέλεση του δεύτερου μέρους του αλγορίθμου ώστε να αφαιρεθούν αυτά τα γνήσια υποσύνολα και να προκύψουν τα παρακάτω υπερσύνολα:

2,5	0,1,2,3,4	7,8	5,6,7	11,13	3,26
16,17,18,13,14,15	16,19	24,25,26,27	19,20,21,22,23	23,24	8,9,10,11,12

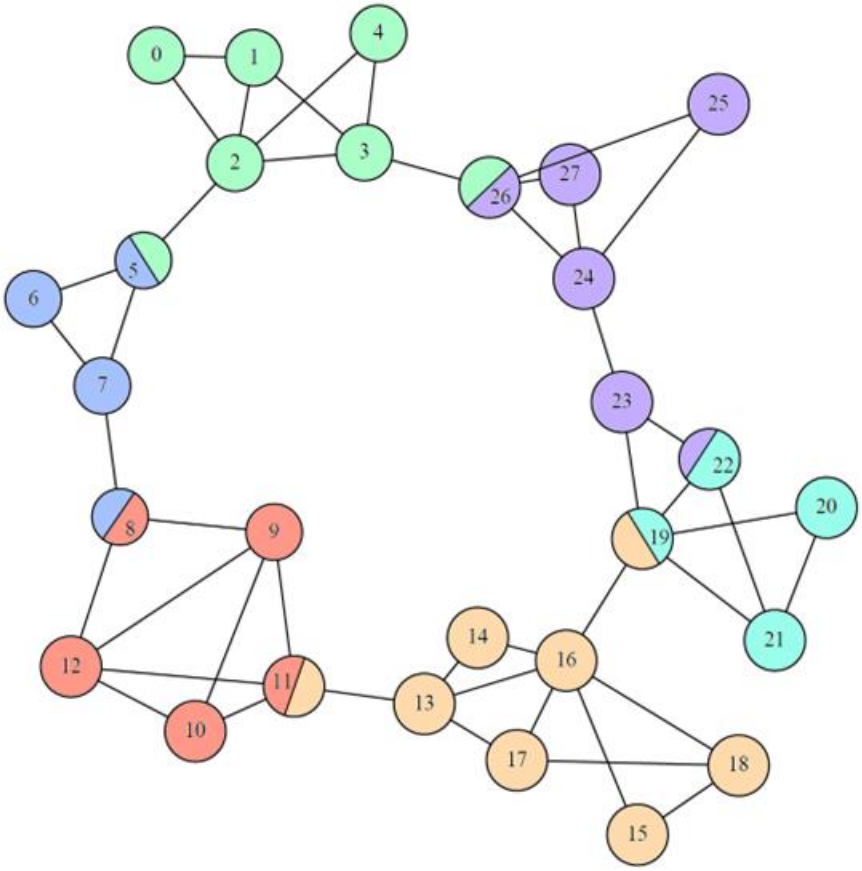
Πίνακας 11: Υπερσύνολα Δεύτερου Παραδείγματος

Αναμενόμενα, προκύπτουν δώδεκα υπερσύνολα. Τέλος, εκτελείται και το τελευταίο τμήμα της ένωσης «κοντινών» συνόλων ώστε να προκύψουν οι παρακάτω τελικές κοινότητες του γράφου:

5,6,7,8	0,1,2,3,4,5,26	16,17,18,119,11,13,14,15	23,24,25,26,27	19,20,21,22,23
8,9,10,11,12				

Πίνακας 12: Λίστα Κοινοτήτων Δεύτερου Παραδείγματος

Παρατηρούμε ότι προέκυψαν έξι κοινότητες στον γράφο που χρησιμοποιήθηκε αυτή την φορά, ενώ υπάρχει επικάλυψη κοινοτήτων στις κορυφές 5, 8, 11, 19, 22 και 26. Ο γράφος χωρισμένος σε κοινότητες όπως προέκυψαν από την εκτέλεση του αλγορίθμου παρουσιάζεται στην παρακάτω εικόνα:



Εικόνα 24: Απεικόνιση Κοινοτήτων Δεύτερου Παραδείγματος

Κεφάλαιο 5ο Συμπεράσματα

Ανακεφαλαιώνοντας, σε προβλήματα που συναντάμε καθημερινά σε διάφορους τύπους επιστημονικών κλάδων υπάρχει η δυνατότητα αναπαράστασης προβλημάτων αλλά και καταστάσεων σε μορφή δικτύων τα οποία στην συνέχεια μπορούν να αναλυθούν και να επεξεργαστούν κατάλληλα, παρέχοντας νέα δεδομένα και βοηθώντας την εξαγωγή συμπερασμάτων.

Στα πλαίσια της παρούσας πτυχιακής εργασίας υλοποιήθηκε ένας αλγόριθμος εύρεσης επικαλυπτόμενων κοινοτήτων σε γράφους με την βοήθεια πόρων από την βιβλιοθήκη ανοιχτού κώδικα JGraphT. Η υλοποίηση χωρίστηκε σε δύο βασικά μέρη, αυτό της εύρεσης του ego network των κορυφών ενός γράφου, που δίνει μια εγωκεντρική άποψη στην εύρεση κοινοτήτων, ενώ το δεύτερο στην υλοποίηση του κύριου μέρους του αλγορίθμου όπως αυτός περιγράφεται από τους αναλυτές Bradley S. Rees και Keith B. Gallagher.

Κατά την ανάπτυξη του προγράμματος έγιναν διάφορες τροποποιήσεις στην διαδικασία του αλγορίθμου, όπως για παράδειγμα ο «άπληστος» τρόπος αναζήτησης των συνόλων που είναι «κοντά» σε άλλα σύνολα. Επιπλέον εισήχθη ένας μηχανισμός που επιτρέπει στον χρήστη του αλγορίθμου να εκτελεί μέρος του αλγορίθμου με την χρήση νημάτων, κάτι που βελτιώνει αισθητά τον χρόνο εκτέλεσης του, ιδιαίτερα όταν αναφερόμαστε σε μεγάλους γράφους και σε μηχανήματα που διαθέτουν ικανοποιητικό αριθμό νημάτων προς χρήση.

Η πρακτική υλοποίηση του αλγορίθμου θα μπορούσε με τα κατάλληλα βήματα να βελτιωθεί μιας και υπάρχουν ελλείψεις και επιπλέον δυνατότητες βελτίωσης της χρονικής απόδοσης του. Οι βασικές βελτιώσεις που θα μπορούσαν να πραγματοποιηθούν είναι οι παρακάτω:

- Τροποποίηση του τμήματος ένωσης «κοντινών» συνόλων ώστε να μπορεί να εκτελεστεί ευκολότερα σε παράλληλη λογική.
- Χρήση νημάτων που προσφέρουν οι μονάδες επεξεργασίας γραφικών λόγω του μεγάλου αριθμού νημάτων που μπορούν να χρησιμοποιηθούν ταυτοχρόνως, μέσω μίας διεπαφής του τύπου OpenCL.

Μελλοντικός στόχος για την συγκεκριμένη υλοποίηση που αναπτύχθηκε είναι να τροποποιηθεί σε συγκεκριμένα σημεία με στόχο να τηρεί τα πρότυπα της βιβλιοθήκης JGraphT και τελικά να προστεθεί σε αυτήν στο ειδικό τμήμα υλοποιημένων αλγορίθμων εύρεσης κοινοτήτων σε γράφους που περιέχει.

Βιβλιογραφία

Papers

[1] Freeman, Linton C., “Centered graphs and the structure of ego networks”, Mathematical Social Sciences, North-Holland Publishing Company, vol. 3, issue 3, pages 291-304, 1982.

[2] Bradley S. Rees and Keith B. Gallagher, “Overlapping Community Detection by Collective Friendship Group Inference”, 2010 International Conference on Advances in Social Network Analysis and Mining, pages 375-379, 2010.

[3] Dimitrios Michail, Joris Kinable, Barak Naveh and John V. Sichi, “JGraphT – A java Library for Graph Data Structures and Algorithms”, ACM Transactions on Mathematical Software, vol. 46, issue 2, pages 1-29, 2020.

[4] Girvan M., Newman M.E., “Community structure in social and biological networks”, Proc. Natl. Acad. Sci. USA, 2002.

[5] Girvan M., “Modularity and community structure in networks, Proc. Natl. Acad. Sci. USA, 2006.

[6] Blondel, Vincent & Guillaume, Jean-Loup & Lambiotte, Renaud & Lefebvre, Etienne, “Fast Unfolding of Communities in Large Networks”, Journal of Statistical Mechanics Theory and Experiment, 2008.

[7] V. Arnaboldi, M. Conti, A. Passarella and F. Pezzoni, “Analysis of Ego Network Structure in Online Social Networks”, 2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing, pages 31-40, 2012.

[8] R. DeJordy and D. Halgin, “Introduction into Ego Network Analysis.”, 2009.

Websites

[9] “Graph (discrete mathematics).”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Graph_(discrete_mathematics) (accessed Aug. 1, 2022).

[10] “Community Structure.”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Community_structure (accessed Aug. 1, 2022).

[11] “Graph Theory.”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Graph_theory (accessed Aug. 1, 2022).

[12] “Community Structure.”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Community_structure (accessed Aug. 1, 2022).

[13] “Girvan Newman Algorithm.”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Girvan%E2%80%93Newman_algorithm (accessed Aug. 1, 2022).

[14] “Modularity (Networks).”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Modularity_(networks) (accessed Aug. 1, 2022).

[15] “Louvain Method.”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Louvain_method (accessed Aug. 1, 2022).

[16] “Breadth-first Search.”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Breadth-first_search (accessed Aug. 1, 2022).

[17] “Dijkstra’s algorithm.”, Wikipedia: The Free Encyclopedia, Wikimedia Foundation. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm (accessed Aug. 1, 2022).