ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

“ΑΠΟΣΥΝΘΕΣΗ ΠΥΡΗΝΑ ΚΑΙ ΕΥΡΕΣΗ ΤΟΥ ΠΥΚΝΟΤΕΡΟΥ ΥΠΟΓΡΑΦΟΥ ΣΕ ΠΟΛΥΕΠΙΠΕΔΟΥΣ ΓΡΑΦΟΥΣ”

ΚΟΥΒΕΛΑ ΜΑΡΙΑ
ΑΕΜ: 2360

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ:
ΠΑΠΑΔΟΠΟΥΛΟΣ ΑΠΟΣΤΟΛΟΣ, ΕΠΙΚΟΥΡΟΣ ΚΑΘΗΓΗΤΗΣ

ΘΕΣΣΑΛΟΝΙΚΗ, ΦΕΒΡΟΥΑΡΙΟΣ 2018

ΕΥΧΑΡΙΣΤΙΕΣ

Με την περάτωση της πτυχιακής μου εργασίας θα ήθελα αρχικά να ευχαριστήσω τον Ναπολέων, τον άνθρωπο που ήταν δίπλα μου καθ’ όλη την διάρκεια των σπουδών μου, που είχε πάντα την θέληση να με βοηθήσει και που μου παρείχε την ψυχολογική υποστήριξη ώστε να συνεχίζω να αγωνίζομαι. Επίσης θέλω να ευχαριστήσω την οικογένειά μου, που με στήριξε με κάθε θυσία και χωρίς την οποία δεν θα ήμουν σήμερα εδώ. Τέλος, θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Απόστολο Παπαδόπουλο για την άριστη συνεργασία μας και την εμπιστοσύνη του σε μένα.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΕΦΑΛΑΙΟ 1:
ΠΕΡΙΛΗΨΗ - ΕΙΣΑΓΩΓΗ

ΠΕΡΙΛΗΨΗ

Αντικείμενο της παρούσας εργασίας είναι ο υπολογισμός της αποσύνθεσης πυρήνα σε πολυεπίπεδο γράφημα και χρήση των πυρήνων του για την εύρεση του πυκνότερου υπογράφου. Το πρόβλημα του υπολογισμού της αποσύνθεσης πυρήνα σε ένα πολυεπίπεδο γράφημα παρουσιάζει μεγάλο ενδιαφέρον στην μελέτη κοινωνικών δικτύων (Facebook, Twitter κλπ.), καθώς και σε βιολογικά και οικονομικά δίκτυα. Από την άλλη μεριά, ένα κλασικό πρόβλημα της θεωρίας γράφων είναι η εύρεση του πυκνότερου υπογράφου ενός γράφου, για το οποίο χρησιμοποιείται μια νέα προσέγγιση. Στην εργασία μελετώνται δύο αλγόριθμοι για την εύρεση των πυρήνων σε πολυεπίπεδα δίκτυα, ο BruteForce (αλγόριθμος ωμής βίας) και ο BFS (αλγόριθμος πρώτα κατά πλάτος), και ένας αλγόριθμος για την εύρεση του πυκνότερου υπογράφου, ο DensestSubgraph.

ABSTRACT

The main concept of this work is computing the core decomposition of a multilayer graph and using its cores to find the densest subgraph. Computing the core decomposition of a multilayer graph is attracting interest in studying social networks (Facebook, Twitter etc.), as well as in several other real- world contexts such as biological and financial networks. On the other hand, a classic graph - theory problem is extracting the densest subgraph of a graph, for which it is provided a new point of view. In this work, two algorithms are being studied for finding the cores of multilayer networks, the BruteForce and the BFS algorithm, and one for extracting the densest subgraph, the DensestSubgraph algorithm.

ΕΙΣΑΓΩΓΗ

Ένα γράφημα ή ένας γράφος είναι μια αφηρημένη αναπαράσταση ενός συνόλου στοιχείων, όπου μερικά ζευγάρια στοιχείων συνδέονται μεταξύ τους με δεσμούς. Τα διασυνδεδεμένα στοιχεία αναπαριστώνται με μαθηματικές έννοιες οι οποίες ονομάζονται κορυφές ενώ οι δεσμοί που συνδέουν τα ζευγάρια των κορυφών ονομάζονται ακμές. Συνήθως, ένας γράφος απεικονίζεται σε διαγραμματική μορφή ως ένα σύνολο κουκκίδων για τις κορυφές, ενωμένα μεταξύ τους με γραμμές ή καμπύλες για τις ακμές.

Στην πιο κοινή έννοια του όρου, ένας γράφος είναι ένα διατεταγμένο ζεύγος G = (V, E) αποτελούμενο από ένα σύνολο V των κορυφών ή κόμβων μαζί με E σύνολο από ακμές ή γραμμές, οι οποίες είναι υποσύνολα δύο στοιχείων V (δηλαδή μία ακμή σχετίζεται με δύο κορυφές και η σχέση απεικονίζεται ως μη ταξινομημένο ζεύγος των κορυφών σε σχέση με τη συγκεκριμένη τιμή).

Στην παρούσα εργασία, όταν αναφερόμαστε στην έννοια γράφο εννοούμε έναν απλό, μη κατευθυνόμενο γράφο. Ένας γράφος είναι απλός όταν δεν περιέχει παράλληλες ακμές και βρόχους, δηλαδή ακμές οι οποίες αρχίζουν και τελειώνουν σε ίδια κορυφή. Ένας μη κατευθυνόμενος γράφος είναι εκείνος στον οποίο οι ακμές δεν έχουν προσανατολισμό, όπου δηλαδή η ακμή (i, j) είναι ταυτόσημη με την ακμή (j, i).

Μία σημαντική έννοια που θα χρησιμοποιηθεί αρκετά στην πορεία του κειμένου είναι ο βαθμός μιας κορυφής. Ο βαθμός μιας κορυφής v του γράφου είναι ο αριθμός των ακμών που προσπίπτουν σε αυτήν και συμβολίζεται με d(v) ή deg(v).

Ένα πολυεπίπεδο γράφημα G για ένα σύνολο επιπέδων L = {1, ..., l} είναι ένα σύνολο G = {Gi | i ∈ L} από γραφήματα.

$$Gi = (V, Ei), Ei \subseteq V \times V : Ei \rightarrow R$$

όπου κάθε επίπεδο του γραφήματος Gi , i ∈ Dim είναι ένα μη κατευθυνόμενο γράφημα χωρίς βρόχους. Πιο απλά, ένας πολυεπίπεδος γράφος L επιπέδων αποτελείται από ένα σύνολο L απλών μη κατευθυνόμενων γράφων, όπου το σύνολο κορυφών V είναι το ίδιο για κάθε γράφο, ενώ το σύνολο κορυφών E είναι αυτό που διαφέρει σε κάθε γράφο.

Παραδείγματα πολυεπίπεδων γράφων αποτελούν τα κοινωνικά δίκτυα (social networks). Τα διαφορετικά επίπεδα αντιπροσωπεύουν τα διάφορα κοινωνικά δίκτυα (facebook, instagram, twitter κτλ.). Οι κορυφές των γράφων αντιπροσωπεύουν τα προφίλ των χρηστών, ενώ οι ακμές που ενώνουν τις κορυφές μεταξύ τους αντιπροσωπεύουν τη σύνδεση- φιλία μεταξύ των χρηστών. Οι ακμές σε κάθε γράφο διαφέρουν γιατί, για παράδειγμα, μπορεί ένας χρήστης x να είναι φίλος με τον y σε ένα δίκτυο, ενώ σε ένα άλλο δίκτυο δεν υπάρχει αυτή η σύνδεση.

Η παρούσα εργασία βασίζεται στην εργασία των Edoardo Galimberti, Francesco Bonchi και Francesco Gullo, με τίτλο “Core Decomposition and Densest Subgraph in Multilayer Networks”, και επιχειρεί να υλοποιήσει αλγορίθμους που περιγράφονται σε αυτό.

Οι αλγόριθμοι της εργασίας αναπτύχθηκαν σε γλώσσα Java και επίσης χρησιμοποιήθηκε η δωρεάν βιβλιοθήκη JGraphT, η οποία παρέχει υλοποιημένες κλάσεις και αλγόριθμους και υποστηρίζει διάφορα είδη γράφων.

Η μελέτη των πολυεπίπεδων γράφων, οι οποίοι αναπαριστούν κοινωνικά δίκτυα, αποδεικνύεται χρήσιμη σε τομείς όπως οι επιχειρήσεις, όπου για παράδειγμα έχει νόημα η εύρεση και προσέγγιση μιας κοινότητας, ενός συνόλου χρηστών με κοινά χαρακτηριστικά, με σκοπό τη διαφήμιση κάποιου προϊόντος/ υπηρεσίας.

Η εύρεση τέτοιων συνόλων χρηστών ονομάζεται αποσύνθεση πυρήνα, όπου κάθε σύνολο αποτελεί ένα πυρήνα. Στην εργασία αυτή πραγματοποιείται ανάλυση δύο αλγορίθμων υπολογισμού της αποσύνθεσης πυρήνα σε πολυεπίπεδους γράφου, ο BruteForce, αλγόριθμος ωμής βίας και ο BFS ο οποίος εκμεταλλεύεται κάποιους αποτελεσματικούς κανόνες κλαδέματος του πλέγματος - χώρου αναζήτησης.

Μία από τις πιο ενδιαφέρουσες ιδιότητες ενός γράφου είναι η πυκνότητά του, δηλαδή το πόσο στενά συνδεδεμένα είναι τα διάφορα τμήματά του μεταξύ τους. Στην παρούσα εργασία γίνεται μελέτη ενός αλγορίθμου εξαγωγής του πυκνότερου υπογράφου ενός πολυεπίπεδου γράφου, του αλγορίθμου DensestSubgraph, ο οποίος χρησιμοποιεί τους πυρήνες του γράφου για τους υπολογισμούς του.

Η εργασία δομείται σε κεφάλαια ως εξής:

Στο Κεφάλαιο 2 παρουσιάζεται και αναλύεται ο αλγόριθμος BruteForce για τον υπολογισμό της αποσύνθεσης πυρήνα ενός γράφου.

Στο Κεφάλαιο 3 παρουσιάζεται και αναλύεται ο αλγόριθμος BFS για το υπολογισμό της αποσύνθεσης του πυρήνα ενός γράφου.

Στο Κεφάλαιο 4 παρουσιάζεται και αναλύεται ο αλγόριθμος DensestSubgraph για την εύρεση του πυκνότερου υπογράφου ενός γράφου βάση των πυρήνων του.

Στο Κεφάλαιο 5 παρουσιάζονται μετρήσεις και χρονικές αποδόσεις των δύο αλγορίθμων υπολογισμού αποσύνθεσης πυρήνα.

Στο Κεφάλαιο 6 γίνεται μία σύνοψη της εργασίας και αναφέρονται θέματα για περαιτέρω μελέτη.

ΚΕΦΑΛΑΙΟ 2:
ΑΛΓΟΡΙΘΜΟΣ BRUTE FORCE

ΚΕΦΑΛΑΙΟ 2: ΑΛΓΟΡΙΘΜΟΣ BRUTE FORCE

2.1 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Ορισμός Αποσύνθεση πυρήνα σε έναν απλό γράφο ενός επιπέδου
Σε ένα γράφο G = (V, E), για κάθε κορυφή v ∈ V, ο βαθμός της κορυφής v συμβολίζεται deg(v) και $deg_S(v)$, στο γράφο G και σε ένα υπογράφο S του G αντίστοιχα.
Ο πυρήνας k-core (τάξης k) του γράφου G είναι ένας υπογράφος S = (C_k, E[C_k]) όπου κάθε κορυφη u ∈ C_k έχει βαθμό $deg_S(u) ≥ k$. Το σύνολο όλων των πυρήνων k - cores αποτελεί την αποσύνθεση πυρήνα (core decomposition) του γράφου G.

Στην περίπτωση ενός πολυεπίπεδου γράφου *G = (V, E, L)*, όπου V είναι ένα σύνολο κορυφών, L είναι ένα σύνολο επιπέδων και E ⊆ V × V × L ένα σύνολο ακμών, η διαδικασία εύρεσης της αποσύνθεσης πυρήνα είναι λίγο διαφορετική. Δοθέντος ενός |L|-διάστατου διανύσματος ακεραίων $\boldsymbol{k} = [k_\ell]_{\ell \in L}$, το οποίο ονομάζεται coreness vector, ο πολυεπίπεδος k-core πυρήνας του G είναι ο μέγιστος υπογράφος του οποίου οι κορυφές έχουν τουλάχιστον βαθμό k_ℓ σε αυτό τον υπογράφο, για όλα τα επίπεδα ℓ. Το σύνολο όλων των διακριτών και μη κενών πολυεπίπεδων πυρήνων αποτελεί την πολυεπίπεδη αποσύνθεση πυρήνα (multilayer core decomposition). Μία παρατήρηση είναι ότι ένας πολυεπίπεδος πυρήνας μπορεί να περιγραφεί με παραπάνω από ένα διάνυσμα k.

Η λογική στον υπολογισμό για την εύρεση του multilayer core decomposition είναι η εξής: Για κάθε δυνατό διάνυσμα k, για το αντίστοιχο k_ℓ του επιπέδου ℓ αναζητούνται οι κορυφές που στο επίπεδο αυτό έχουν βαθμό μεγαλύτερο ή ίσο του k_ℓ. Η διαδικασία αυτή πραγματοποιείται για όλα τα k_ℓ του k, και στόχος είναι η εύρεση των κοινών κορυφών από το σύνολο των διαφορετικών k_ℓ που ανήκουν στο ίδιο k. Το κάθε σύνολο των κορυφών που σχηματίζεται αποτελεί έναν πυρήνα του γράφου, και όλα τα διαφορετικά και ταυτόχρονα μη κενά σύνολα αποτελούν την πολυεπίπεδη αποσύνθεση του πυρήνα. Η μέθοδος αυτή απαιτεί τον εκ τον προτέρων υπολογισμό όλων των διανυσμάτων $k = [k_\ell]_{\ell \in L}$, όπου όμως δεν υπάρχει αρχική γνώση για το μέγιστο βαθμό που μπορεί να πάρει κάθε τιμή k_ℓ. Μία λύση για την καταμέτρηση όλων των δυνατών διανυσμάτων k είναι η εύρεση του μέγιστου βαθμού maxD στον αρχικό γράφο. Έτσι κάθε k_ℓ παίρνει τιμές στο [0, maxD].

Input: A multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, an $|L|$-dimensional integer vector $\mathbf{k} = [k_\ell]_{\ell \in L}$.
Output: The $\mathbf{k}$-core $C_{\mathbf{k}}$ of G.
1: **while** $\exists u \in S, \exists \ell \in L : deg_S(u, \ell) < k_\ell$ **do**
2: $S \leftarrow S \setminus \{u\}$
3: $C_{\mathbf{k}} = S$

Σχήμα 1: Ψευδοκώδικας του αλγορίθμου BruteForce
(Πηγή: "Core Decomposition and Densest Subgraph in Multilayer Networks", 2017)

Μια αδυναμία που παρουσιάζεται στη λογική του αλγορίθμου ωμής βίας είναι ότι, αφενός, κάθε πυρήνας υπολογίζεται από τον αρχικό γράφο, και, αφετέρου, μπορεί να προκύψουν αρκετοί ίδιοι ή κενοί πυρήνες.

2.2 ΠΡΟΕΤΟΙΜΑΣΙΑ ΑΡΧΕΙΟΥ ΕΙΣΟΔΟΥ

Για την αναπαράσταση των γράφων στον κώδικα δημιουργήθηκαν αρχεία .txt. Κάθε τέτοιο αρχείο ξεκινά με την πρώτη γραμμή να περιέχει τα δεδομένα του γράφου στη μορφή “πλήθος - επιπέδων πλήθος - κορυφών πλήθος - κορυφών”. Στη συνέχεια κάθε γραμμή περιέχει την πληροφορία για το ποια ακμή περιέχεται σε ποιο επίπεδο με τη μορφή “επίπεδο κορυφή1 κορυφή2” όπου η κορυφή1 συνδέεται με ακμή με την κορυφή2.

public static Multigraph<String, GraphLayerEdge> createMultigraph(String dataset)

Όλοι οι αλγόριθμοι που αναπτύχθηκαν στις αντίστοιχες κλάσεις, αρχικά καλούν τη συνάρτηση *createMultigraph( )*, η οποία δέχεται σαν όρισμα το όνομα του αρχείου .txt, ώστε να δημιουργηθεί ο γράφος προς επεξεργασία. Η *createMultigraph( )* δημιουργεί ένα γράφο - αντικείμενο *Multigraph<String, GraphLayerEdge>*, έπειτα διαβάζει το αρχείο που δόθηκε σαν όρισμα γραμμή- γραμμή και προσθέτει στο γράφο τις κορυφές και τις αντίστοιχες ακμές σε κάθε επίπεδο, ελέγχοντας αν αυτές υπάρχουν ήδη, και τέλος επιστρέφει το γράφο. Επειδή δεν υπάρχει κάποιος τρόπος αναφοράς στις ακμές του γράφου, αυτές κατά τη δημιουργία τους δέχονται το όνομα του επιπέδου ως *ετικέτα* (label) μέσω της κλάσης *GraphLayerEdge*.

2.3 ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΟΥ ΣΕ ΚΩΔΙΚΑ JAVA

Για την υλοποίηση του αλγορίθμου BruteForce δημιουργήθηκαν τέσσερις συναρτήσεις που φαίνονται παρακάτω:

public static void main(String[ ] args)

Αρχικά καλείται η συνάρτηση createMultigraph( ) με όρισμα το όνομα του αρχείου από το οποίο θα διαβαστεί και θα δημιουργηθεί ο πολυεπίπεδος γράφος. Στη συνέχεια δημιουργείται ένας δισδιάστος πίνακας ακεραίων degree[ ][ ] θέσεων μήκους πλήθος επιπέδων x πλήθος κορυφών του γράφου, και καλείται η συνάρτηση findDegree( ) ώστε να γεμίσει τις θέσεις του πίνακα. Έπειτα γίνεται αναζήτηση στον πίνακα για τη μέγιστη τιμή του, η οποία ανατίθεται στη μεταβλητή maxD. Τέλος, καλείται η συνάρτηση findCoreDecomposition( ) με όρισμα τη μεταβλητή maxD.

private static void findDegree(Multigraph<String, GraphLayerEdge> tempG)

Αρχικοποιείται ο πίνακας degree[ ][ ] με μηδενικά. Έπειτα για κάθε κορυφή i του γράφου δημιουργείται ένα σύνολο ακμών που προσπίπτουν σε αυτήν. Για κάθε ακμή του συνόλου αυτού επιστρέφεται η ετικέτα (label) l αυτής που αντιπροσωπεύει το επίπεδο στο οποίο υπάρχει η ακμή. Οπότε η θέση degree[l][i] αυξάνεται κατά ένα.

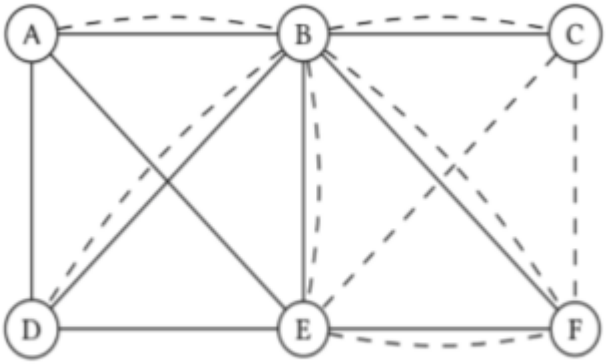
private static Multigraph<String, GraphLayerEdge> updateGraph(Multigraph<String, GraphLayerEdge> g, int layer, int vertex)

Η συνάρτηση αυτή δέχεται σαν όρισμα έναν γράφο g, ένα επίπεδο layer και μια κορυφή vertex. Για κάθε ακμή (edge) που υπάρχει στο σύνολο κορυφών του γράφου g, αν η ετικέτα (label) της ακμής είναι η τιμή του layer και ταυτοχρόνως η ακμή προσπίπτει στην κορυφή vertex, τότε η ακμή αυτή διαγράφεται από το σύνολο ακμών του γράφου. Όταν πραγματοποιηθούν όλες οι αναγκαίες διαγραφές, επιστρέφεται ο ανανεωμένος γράφος g.

private static void findCoreDecomposition (int maxD)

Η συνάρτηση δέχεται σαν όρισμα την τιμή maxD, την οποία χρησιμοποιεί αρχικά για να κατασκευάσει μια λίστα από string μονοδιάστατους πίνακες μήκους όσο το πλήθος των επιπέδων, την all_Ks. Οι πίνακες της λίστας περιέχουν στοιχεία k_ϱ τα οποία παίρνουν τιμές στο [0, maxD]. Αφού αρχικοποιηθεί η all_Ks, δημιουργείται μία νέα λίστα από λίστες ακεραίων, η coreDecompositionOfK, στην οποία θα προστεθούν οι πυρήνες του γράφου, όπου κάθε πυρήνας θα αποτελείται από μία λίστα κορυφών. Στην συνέχεια, για κάθε πίνακα k[ ] της all_Ks δημιουργείται ένα αντίγραφο του αρχικού γράφου, ώστε να μη μεταβληθεί ο αρχικός κατά τη διαγραφή ακμών, ο tempMg, και μια λίστα ακεραίων verticesSet η οποία αρχικοποιείται με τις κορυφές του tempMg, και καλείται η findDegree( ) για την αρχικοποίηση του πίνακα degree[ ][ ]. Έπειτα για κάθε κορυφή της verticesSet και για κάθε επίπεδο l του αρχικού γράφου, ορίζεται η μεταβλητή kl η οποία παίρνει την τιμή του l στοιχείου του πίνακα k[ ]. Εάν η τιμή degree[l][v], δηλαδή ο βαθμός της κορυφής v στο επίπεδο l, είναι μικρότερη της τιμής του kl, τότε διαγράφεται η κορυφή v από τη λίστα verticesSet, αφού δεν μπορεί να περιέχεται στον πυρήνα. Καλείται η updateGraph( ), με ορίσματα τον tempMg, το επίπεδο l και την κορυφή v, και η findDegree( ) με όρισμα τον tempMg ώστε να ανανεωθούν οι βαθμοί των κορυφών στον πίνακα degree[ ][ ]. Τέλος κάθε λίστα verticesSet η οποία έχει μέγεθος μεγαλύτερο του ένα προστίθεται στη λίστα coreDecompositionOfK, εάν δεν περιέχεται ήδη σε αυτή. Εκτυπώνονται η αποσύνθεση πυρήνα, το πλήθος των υπολογισμένων πυρήνων και το πλήθος των διακριτών και μη κενών πυρήνων.

2.4 ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ



Σχήμα 2: Παράδειγμα γράφου δύο επιπέδων
(Πηγή: "Core Decomposition and Densest Subgraph in Multilayer Networks", 2017)

Στο σχήμα 2 φαίνεται ένας πολυεπίπεδος γράφος με έξι κορυφές και δύο επίπεδα. Θα υπολογιστεί η αποσύνθεση του πυρήνα με τον αλγόριθμο Brute Force.

Θεωρώντας ότι έχει κατασκευαστεί ο πίνακας degree[ ][ ] βρίσκουμε το maxD να ισούται με πέντε. Οπότε τα kl θα περιέχουν δύο στοιχεία τα οποία θα παίρνουν τιμές στο [0, 5] και σύμφωνα με την συνάρτηση findCoreDecomposition θα γίνουν οι εξής υπολογισμοί:

k = [0, 0]	verticesSet = [A, B, C, D, E, F]	k = [3, 0]	verticesSet = [A, B, D, E]
k = [0, 1]	verticesSet = [A, B, C, D, E, F]	k = [3, 1]	verticesSet = [A, B, D, E]
k = [0, 2]	verticesSet = [B, C, E, F]	k = [3, 2]	verticesSet = [ ]
k = [0, 3]	verticesSet = [B, C, E, F]	k = [3, 3]	verticesSet = [ ]
k = [0, 4]	verticesSet = [ ]	k = [3, 4]	verticesSet = [ ]
k = [0, 5]	verticesSet = [ ]	k = [3, 5]	verticesSet = [ ]
k = [1, 0]	verticesSet = [A, B, C, D, E, F]	k = [4, 0]	verticesSet = [ ]
k = [1, 1]	verticesSet = [A, B, C, D, E, F]	k = [4, 1]	verticesSet = [ ]
k = [1, 2]	verticesSet = [B, C, E, F]	k = [4, 2]	verticesSet = [ ]
k = [1, 3]	verticesSet = [B, C, E, F]	k = [4, 3]	verticesSet = [ ]
k = [1, 4]	verticesSet = [ ]	k = [4, 4]	verticesSet = [ ]
k = [1, 5]	verticesSet = [ ]	k = [4, 5]	verticesSet = [ ]
k = [2, 0]	verticesSet = [A, B, D, E, F]	k = [5, 0]	verticesSet = [ ]
k = [2, 1]	verticesSet = [A, B, D, E, F]	k = [5, 1]	verticesSet = [ ]
k = [2, 2]	verticesSet = [B, E, F]	k = [5, 2]	verticesSet = [ ]
k = [2, 3]	verticesSet = [ ]	k = [5, 3]	verticesSet = [ ]
k = [2, 4]	verticesSet = [ ]	k = [5, 4]	verticesSet = [ ]
k = [2, 5]	verticesSet = [ ]	k = [5, 5]	verticesSet = [ ]

The core decomposition is: [[A, B, C, D, E, F], [B, C, E, F], [A, B, D, E, F], [B, E, F], [A, B, D, E]]
Number of computed cores is: 35
Number of cores is: 5

Παρατηρούμε ότι υπάρχουν αρκετοί κενοί πυρήνες.

ΚΕΦΑΛΑΙΟ 3:
ΑΛΓΟΡΙΘΜΟΣ BFS

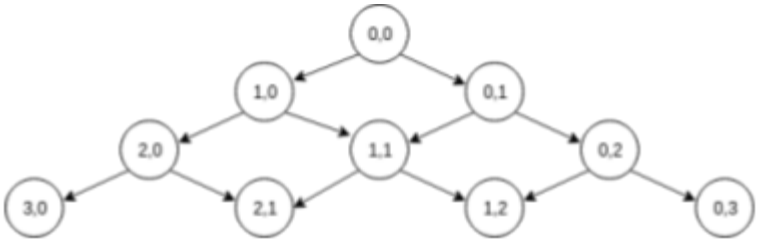
ΚΕΦΑΛΑΙΟ 3: ΑΛΓΟΡΙΘΜΟΣ BFS

3.1 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Λόγω της σύμβασης που κάνει ο αλγόριθμος BruteForce σχετικά με την εκ των προτέρων καταμέτρηση των διανυσμάτων k, η πολυπλοκότητά του είναι της τάξης $O((|E|+|V|\times|L|)\times\Pi_{\ell\in L}k_\ell)$.

Παρ' ότι οι πυρήνες δεν είναι φωλιασμένοι μεταξύ τους, υπάρχει μία σχέση ανάμεσά τους. Συγκεκριμένα δοθέντος ενός πολυεπίπεδου γράφου G = (V, E, L) και δύο πυρήνων C_k και $C_{k'}$ του G με διανύσματα $k = [k_\ell]_{\ell\in L}$ και $k' = [k'_\ell]_{\ell\in L}$ αντίστοιχα, ισχύει ότι αν $\forall \ell \in L: k'_\ell \le k_\ell$, τότε $C_k \subseteq C_{k'}$ *(Σχέση 1).*

Με βάση αυτή τη σχέση ο χώρος αναζήτησης του προβλήματος μπορεί να αναπαρασταθεί σαν πλέγμα ορίζοντας μια μερική σειρά μεταξύ των πυρήνων. Το πλέγμα αυτό καλείται πλέγμα πυρήνων και αντιστοιχεί σε έναν κατευθυνόμενο ακυκλικό γράφο όπου οι κόμβοι αναπαριστούν τους πυρήνες και οι συνδέσεις τις μεταξύ τους σχέσεις (ένας πυρήνας - πατέρας περιέχει όλους τους πυρήνες - παιδιά του). Ένας πυρήνας υπάρχει στο πλέγμα τόσες φορές όσες ο αριθμός των διανυσμάτων. Σε κάθε επίπεδο i του πλέγματος βρίσκονται τα παιδιά των πυρήνων του επιπέδου i - 1. Αναλυτικότερα, το επίπεδο i περιέχει όλους τους πυρήνες των οποίων τα διανύσματα k προκύπτουν από την αύξηση κατά ένα μίας μόνο συντεταγμένης του διανύσματος του πατέρα, με εξαίρεση το επίπεδο 0 που περιέχει μόνο τη ρίζα, δηλαδή όλες τις κορυφές του γράφου.



Σχήμα 3: Πλέγμα πυρήνα γράφου δύο επιπέδων

Μία βελτίωση που προσφέρει ο αλγόριθμος BFS είναι ότι λαμβάνει υπόψιν τη σχέση αυτή και η στρατηγική του ξεκινάει με τη δημιουργία του πλέγματος από τον αρχικό γράφο. Επίσης, από τη σχέση 1 προκύπτουν δύο σημαντικά συμπεράσματα:

1. Δοθέντος ενός πολυεπίπεδου γράφου G, έστω C ένας πυρήνας του γράφου και F(C) το σύνολο των πατέρων του C στο πλέγμα πυρήνων του G. Ισχύει ότι κάθε μη κενός πυρήνας k περιλαμβάνεται υποχρεωτικά στην τομή όλων των κόμβων πατέρων του στο πλέγμα.
2. Δοθέντος ενός πολυεπίπεδου γράφου G, έστω C ένας πυρήνας του γράφου με διάνυσμα $k = [k_\ell]_{\ell\in L}$, και F(C) το σύνολο των πατέρων του C στο πλέγμα πυρήνων του G. Ισχύει ότι κάθε μη κενός πυρήνας k έχει ακριβώς τόσους πατέρες όσο το πλήθος των μη μηδενικών συντεταγμένων του διανύσματός του k.

Η λογική του αλγορίθμου έχει ως εξής: Σε κάθε επανάληψη, εξετάζεται ο πρώτος κόμβος k μιας ουράς (queue), ο οποίος αφαιρείται από αυτή Αν το πλήθος των μη μηδενικών συντεταγμένων του k ισούται με το πλήθος των κόμβων πατέρων του, υπολογίζεται ο πυρήνας του, αλλιώς η επανάληψη προχωράει σε επόμενο διάνυσμα k. Ο υπολογισμός των πυρήνων ξεκινάει από τη ρίζα του πλέγματος, δηλαδή από τον αρχικό γράφο G, ενώ οι υπόλοιποι υπολογίζονται από έναν υπογράφου G, ο οποίος προκύπτει από την τομή των κορυφών του πυρήνα των κόμβων - πατέρων του κόμβου που εξετάζεται κάθε φορά. Με τον τρόπο αυτό αποφεύγεται ο υπολογισμός κενών πυρήνων, καθώς επίσης δεν χρειάζεται να γνωρίζουμε τα διανύσματα k εκ των προτέρων. Αν ο πυρήνας που υπολογίστηκε δεν είναι κενός, προστίθεται στην αποσύνθεση πυρήνα του γράφου, αν δεν περιέχεται ήδη. Έπειτα, προστίθενται οι κόμβοι - παιδιά k' του k στην ουρά queue, και ο πυρήνας του k προστίθεται στο σύνολο πυρήνων των πατέρων των k'.

Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{C}$ of all non-empty multilayer cores of G.

1: $\mathbf{C} \leftarrow \emptyset$, $\mathbf{Q} \leftarrow \{[0]_{|L|}\}$, $\mathcal{F}([0]_{|L|}) \leftarrow \emptyset$ {$\mathcal{F}$ keeps track of father nodes}
2: **while** $\mathbf{Q} \neq \emptyset$ **do**
3: dequeue $\mathbf{k} = [k_\ell]_{\ell \in L}$ from $\mathbf{Q}$
4: **if** $|\{k_\ell : k_\ell > 0\}| = |\mathcal{F}(\mathbf{k})|$ **then** {Corollary 2}
5: $F_\cap \leftarrow \bigcap_{F \in \mathcal{F}(\mathbf{k})} F$ {Corollary 1}
6: $C_\mathbf{k} \leftarrow \mathbf{k}$-core$(G, F_\cap, \mathbf{k})$ {Algorithm 1}
7: **if** $C_\mathbf{k} \neq \emptyset$ **then**
8: $\mathbf{C} \leftarrow \mathbf{C} \cup \{C_\mathbf{k}\}$
9: **for all** $\ell \in L$ **do** {enqueue child nodes}
10: $\mathbf{k}' \leftarrow [k_1, \ldots, k_\ell + 1, \ldots, k_{|L|}]$
11: enqueue $\mathbf{k}'$ into $\mathbf{Q}$
12: $\mathcal{F}(\mathbf{k}') \leftarrow \mathcal{F}(\mathbf{k}') \cup \{C_\mathbf{k}\}$

Σχήμα 4: Ψευδοκώδικας του αλγορίθμου BFS
(Πηγή: "Core Decomposition and Densest Subgraph in Multilayer Networks", 2017)

3.2 ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΟΥ ΣΕ ΚΩΔΙΚΑ JAVA

Για την υλοποίηση του αλγορίθμου BFS δημιουργήθηκαν δέκα συναρτήσεις που φαίνονται παρακάτω:

public static void main(String[ ] args)

Αρχικά καλείται η συνάρτηση createMultigraph( ) με όρισμα το όνομα του αρχείου από το οποίο θα διαβαστεί και θα δημιουργηθεί ο πολυεπίπεδος γράφος. Στη συνέχεια δημιουργείται ένας δισδιάστος πίνακας ακεραίων degree[ ][ ] θέσεων μήκους πλήθος επιπέδων x πλήθος κορυφών του γράφου. Τέλος, καλείται η συνάρτηση findCoreDecomposition( ).

private static ArrayList<ArrayList<Integer>> findCoreDecomposition( )

Αρχικά δημιουργούνται μία λίστα από λίστες, η cores η οποία αντιπροσωπεύει την αποσύνθεση πυρήνα του γράφου, μια λίστα από μονοδιάστατους πίνακες String, η queue η οποία θα λειτουργεί σαν ουρά για τα διάφορα διανύσματα k, ένας μονοδιάστατος πίνακας String k μήκους όσο ο αριθμός των επιπέδων που αντιπροσωπεύει τα διανύσματα k, και ένα Hashmap f το οποίο για κάθε κλειδί - πίνακα k περιέχει μια τιμή - λίστα από τις κορυφές των κόμβων- πατέρων του k. Στην queue προστίθεται το πρώτο k, δηλαδή η ρίζα που αποτελείται από μηδενικά μήκους όσο ο αριθμός των επιπέδων, και στον hm προστίθεται το κλειδί k με τιμή μία κενή λίστα. Στη συνέχεια, όσο η queue δεν είναι κενή, επαναλαμβάνεται η διαδικασία: Το πρώτο διάνυσμα k βγαίνει από την ουρά. Αν το πλήθος των μη μηδενικών συντεταγμένων του k ισούται με το μέγεθος της τιμής - λίστας του Hashmap f με κλειδί τον πίνακα k, τότε ορίζεται μια νέα λίστα, η fIntersection, στην οποία προστίθενται όλες οι κοινές κορυφές που περιέχονται στην τιμή του f(k), δηλαδή οι κορυφές που περιέχονται στον πυρήνα των κόμβων -πατέρων του k στο πλέγμα. Διαφορετικά, η επανάληψη συνεχίζει με το επόμενο διάνυσμα k της queue. Στην περίπτωση της ρίζας k, η fIntersection θα περιέχει όλες τις κορυφές του γράφου. Έπειτα, καλείται η συνάρτηση kCore( ) με ορίσματα την λίστα fIntersection και το διάνυσμα k, και ο πυρήνας που επιστρέφεται προστίθεται σε μία νέα λίστα, την coreDecompositionOfK. Αν η λίστα αυτή δεν είναι κενή, προστίθεται στην λίστα cores αν δεν περιέχεται ήδη σε αυτή. Η διαδικασία συνεχίζεται με την προσθήκη των διανυσμάτων - παιδιών k' του k (αύξηση κατά ένα μιας μόνο συντεταγμένης του k για κάθε επίπεδο) στην queue αν αυτά δεν περιέχονται ήδη σε αυτή, το οποίο ελέγχεται καλώντας τη συνάρτηση containsStringArray με ορίσματα την λίστα queue και τον μονοδιάστατο πίνακα k'[ ] για κάθε παιδί του k. Τέλος, στην τιμή f(k') προστίθεται η λίστα κορυφών του πυρήνα του k, δηλαδή η λίστα coreDecompositionOfK, καλώντας τη συνάρτηση addValue( ) με ορίσματα το Hashmap f, τον πίνακα k' και έναν πίνακα cString, ο οποίος περιέχει τις κορυφές του coreDecompositonOfK. Όταν η queue αδειάσει, εκτυπώνονται μηνύματα σχετικά με τους υπολογισμούς των πυρήνων και επιστρέφεται η λίστα cores, δηλαδή η αποσύνθεση πυρήνα του γράφου, στη συνάρτηση main( ).

private static ArrayList<Integer> kCore(ArrayList<Integer> verticesSet, String[ ] k)

Η συνάρτηση δέχεται σαν όρισμα μία λίστα από κορυφές, την verticesSet, και έναν μονοδιάστατο πίνακα από String, τον k, ο οποίος αντιπροσωπεύει το διάνυσμα k. Αρχικά δημιουργείται μία λίστα, στην οποία θα προστεθούν κορυφές, η coreDecompositionOfK. Δημιουργείται ένα αντίγραφο του αρχικού γραφου, ώστε αυτός να μη μεταβληθεί, ο tempg και καλείται η συνάρτηση findDegree( ), ώστε να αρχικοποιηθεί ο πίνακας degree[ ][ ]. Έπειτα για κάθε κορυφή v του γράφου tempg, αν αυτή δεν υπάρχει στη λίστα verticesSet, τότε για κάθε επίπεδο καλείται η συνάρτηση updateGraph( ) ώστε να διαγραφούν όλες οι ακμές του γράφου που προσπίπτουν στην κορυφή αυτή, καθώς και η updateDegree( ) ώστε να ανανεωθεί ο πίνακας βαθμών degree[ ][ ]. Στη συνέχεια, για κάθε κορυφή της λίστας verticesSet v, και για κάθε επίπεδο l του γράφου, ορίζεται η μεταβλητή kl η οποία έχει τιμή την τιμή της θέσης l στον πίνακα k[ ]. Οπότε, αν ο βαθμός της κορυφής v στο επίπεδο l είναι μικρότερος από την τιμή της kl, τότε η κορυφή v πρέπει να διαγραφεί από τη λίστα verticesSet, αφού δεν μπορεί να περιέχεται στον πυρήνα k. Έτσι, καλείται η updateGraph( ) με ορίσματα tempg, l και v, ώστε να διαγραφούν όλες οι ακμές του tempg που προσπίπτουν στην κορυφή v στο επίπεδο l, αλλά και η updateDegree( ), ώστε να ανανεωθεί ο πίνακας degree[ ][ ]. Μετά το πέρας τον επαναλήψεων, η λίστα coreDecompositionOfK δέχεται όλες τις κορυφές της λίστας verticesSet και επιστρέφεται στο σημείο κλήσης.

private static void findDegree( )

Αρχικοποιείται ο πίνακας degree[ ][ ] με μηδενικά. Έπειτα για κάθε κορυφή i του γράφου δημιουργείται ένα σύνολο ακμών που προσπίπτουν σε αυτήν. Για κάθε ακμή του συνόλου αυτού επιστρέφεται η ετικέτα (label) l αυτής που αντιπροσωπεύει το επίπεδο στο οποίο υπάρχει η ακμή. Οπότε η θέση degree[l][i] αυξάνεται κατά ένα.

private static void updateDegree(Multigraph<String, GraphLayerEdge> tempG, int layer , int vertex)

Η συνάρτηση δέχεται σαν όρισμα έναν πολυεπίπεδο γράφο tempG, ένα επίπεδο layer και μια κορυφή vertex. Αρχικά, ο βαθμός της κορυφής vertex στο επίπεδο πρέπει να μηδενιστεί, οπότε η αντίστοιχη θέση στον πίνακα degree[layer][vertex] παίρνει τιμή μηδέν. Έπειτα για κάθε κορυφή v του συνόλου κορυφών του tempG δημιουργείται ένα σύνολο ακμών το setOfEdges, στο οποίο προστίθενται όλες οι ακμές του tempG που έχουν άκρα τους τις κορυφές v και vertex. Στη συνέχεια, για κάθε ακμή edge του setOfEdges επιστρέφεται η ετικέτα (label) αυτής, και αν αυτή ισούται με την τιμή του επιπέδου layer, τότε η τιμή degree[layer][v] μειώνεται κατά ένα. Δηλαδή για κάθε κορυφή v που συνδέεται με ακμή στο επίπεδο layer με την κορυφή vertex, ο βαθμός της στο επίπεδο layer μειώνεται κατά ένα.

private static Multigraph<String, GraphLayerEdge> updateGraph(Multigraph<String, GraphLayerEdge> g, int layer, int vertex)

Η συνάρτηση αυτή δέχεται σαν όρισμα έναν γράφο g, ένα επίπεδο layer και μια κορυφή vertex. Για κάθε ακμή (edge) που υπάρχει στο σύνολο κορυφών του γράφου g, αν η ετικέτα (label) της ακμής είναι η τιμή του layer και ταυτοχρόνως η ακμή προσπίπτει στην κορυφή vertex, τότε η ακμή αυτή διαγράφεται από το σύνολο ακμών του γράφου. Όταν πραγματοποιηθούν όλες οι αναγκαίες διαγραφές, επιστρέφεται ο ανανεωμένος γράφος g.

private static boolean containsStringArray(List<String[ ]> list, String[ ] aString)

Η συνάρτηση αυτή δέχεται μια λίστα από πίνακες string, τη list, και έναν string μονοδιάστατο πίνακα aString. Για κάθε πίνακα element[ ] της list, αν αυτός ταυτίζεται με τον aString, επιστρέφαται η τιμή true. Αλλιώς, αν τελειώσει η επανάληψη επιστρέφεται η τιμή false.

private static boolean containsKeyArray(HashMap<String[ ], ArrayList<String[ ]>> hm, String[ ] k)

Η συνάρτηση δέχεται ως όρισμα ένα Hashmap hm, με κλειδί ένα μονοδιάστατο πίνακα από String και τιμή μία λίστα από μονοδιάστατους πίνακες String, και έναν μονοδιάστατο πίνακα από String, τον k. Για κάθε κλειδί- πίνακα key του hm, αν ο key ταυτίζεται με τον k, επιστρέφεται η τιμή true. Αλλιώς, αν τελειώσει η επανάληψη επιστρέφεται η τιμή false.

private static ArrayList<String[ ]> getValue(HashMap<String[ ], ArrayList<String[ ]>> hm, String[ ] k)

Η συνάρτηση δέχεται ως όρισμα ένα Hashmap hm, με κλειδί ένα μονοδιάστατο πίνακα από String και τιμή μία λίστα από μονοδιάστατους πίνακες String, και έναν μονοδιάστατο πίνακα από String, τον k. Για κάθε κλειδί- πίνακα key του hm, αν ο key ταυτίζεται με τον k, επιστρέφεται μια λίστα από String με την τιμή (value) hm στη θέση key. Αλλιώς, επιστρέφεται μια κενή λίστα

private static HashMap<String[ ], ArrayList<String[ ]>> addValue(HashMap<String[ ], ArrayList<String[ ]>> hm, String[ ] k, String[ ] value)

Η συνάρτηση δέχεται ως όρισμα ένα Hashmap hm, με κλειδί ένα μονοδιάστατο πίνακα από String και τιμή μία λίστα από μονοδιάστατους πίνακες String, και δύο μονοδιάστατους πίνακες από String, τον k και τον value. Για κάθε κλειδί- πίνακα key του hm, αν ο key ταυτίζεται με τον k, τότε στην τιμή- λίστα του hm στη θέση key προστίθεται ο πίνακας value. Τέλος, επιστρέφεται ο ανανεωμένος hm.

3.3 ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ

Θα χρησιμοποιηθεί το παράδειγμα του σχήματος 2 που χρησιμοποιήθηκε και στην ενότητα 2.4 για την επίδειξη της λειτουργίας του αλγορίθμου BFS. Για κάθε k που προστίθεται στην ουρά queue, επιστρέφεται ο πυρήνας του γράφου

k = [0, 0] coreDecompositionOfK = [A, B, C, D, E, F]
k = [1, 0] coreDecompositionOfK = [A, B, C, D, E, F]
k = [0, 1] coreDecompositionOfK = [A, B, C, D, E, F]
k = [2, 0] coreDecompositionOfK = [A, B, D, E, F]
k = [1, 1] coreDecompositionOfK = [A, B, C, D, E, F]
k = [0, 2] coreDecompositionOfK = [B, C, E, F]
k = [3, 0] coreDecompositionOfK = [A, B, D, E]
k = [2, 1] coreDecompositionOfK = [A, B, D, E, F]
k = [1, 2] coreDecompositionOfK = [B, C, E, F]
k = [0, 3] coreDecompositionOfK = [B, C, E, F]
k = [4, 0] coreDecompositionOfK = [ ]
k = [3, 1] coreDecompositionOfK = [A, B, D, E]
k = [2, 2] coreDecompositionOfK = [B, E, F]
k = [1, 3] coreDecompositionOfK = [B, C, E, F]
k = [0, 4] coreDecompositionOfK = [ ]
k = [4, 1] coreDecompositionOfK δεν υπολογίζεται
k = [3, 2] coreDecompositionOfK = [ ]
k = [2, 3] coreDecompositionOfK = [ ]
k = [1, 4] coreDecompositionOfK δεν υπολογίζεται

The core decomposition is: [[A, B, C, D, E, F], [A, B, D, E, F], [B, C, E, F], [A, B, D, E], [B, E, F]]
Number of computed cores is: 17
Number of cores is: 5

Παρατηρούμε ότι αντίθετα με τον BruteForce υπάρχουν ελάχιστοι κενοί πυρήνες, και επίσης δεν γίνονται οι υπολογισμοί για όλα τα δυνατά k. Παρ’ όλα αυτά τα αποτελέσματα είναι τα ίδια και στους δύο αλγορίθμους.

ΚΕΦΑΛΑΙΟ 4:
ΑΛΓΟΡΙΘΜΟΣ DENSEST SUBGRAPH

ΚΕΦΑΛΑΙΟ 4: ΑΛΓΟΡΙΘΜΟΣ DENSEST SUBGRAPH

4.1 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Για το πρόβλημα της εξαγωγής του πυκνότερου υπογράφου ενός γράφου σε ένα επίπεδο έχουν προταθεί αρκετές λύσεις, ενώ το ίδιο πρόβλημα δεν έχει μελετηθεί αρκετά σε πολυεπίπεδους γράφους. Η εύρεση της αποσύνθεσης πυρήνα σε ένα πολυεπίπεδο γράφο βρίσκει εφαρμογή στο πρόβλημα της εξαγωγής του πυκνότερου υπογράφου, για το οποίο κατασκευάστηκε ένας αλγόριθμος.

Δοθέντος ενός πολυεπίπεδου γράφου G = (V, E, L), η μέση πυκνότητα ενός υποσυνόλου κορυφών S σε ένα επίπεδο ℓ ορίζεται ως το πλήθος των ακμών που προσπίπτουν στις κορυφές του S στο επίπεδο ℓ δια το πλήθος των κορυφών του S. Ο πολυεπίπεδος πυκνότερος υπογράφος ορίζεται ως ένα υποσύνολο κορυφών S^* το οποίο μεγιστοποιεί τη συνάρτηση δ(S) όπου το β είναι μία παράμετρος που ελέγχει τη σημασία της αντιστάθμισης μεταξύ υψηλής πυκνότητας και πλήθους των επιπέδων που εμφανίζουν αυτή την πυκνότητα.

$$\delta(S) = \max_{\hat{L} \subseteq L} \min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|} |\hat{L}|^\beta$$

Η λογική του αλγορίθμου έχει ως εξής: Αρχικά καλείται ένας αλγόριθμος εύρεσης της αποσύνθεσης πυρήνα του πολυεπίπεδου γράφου. Έπειτα είναι στην κρίση του χρήστη να αποφασίσει την τιμή της παραμέτρου β η οποία θα επηρεάσει και το αποτέλεσμα. Στη συνέχεια, για κάθε πυρήνα S του γράφου υπολογίζεται η τιμή της συνάρτησης δ. Ουσιαστικά, για το σύνολο επιπέδων L του γράφου δημιουργείται κάθε δυνατό υποσύνολο $\widehat{L}$ του , και για κάθε στοιχείο ℓ του $\widehat{L}$ αναζητείται η ελάχιστη τιμή του πλήθους ακμών του πυρήνα S στο επίπεδο ℓ ,επί του πλήθους στοιχείων του $\widehat{L}$ υψωμένο στη β, δια το πλήθος κορυφών του πυρήνα S. Αφού βρεθεί η ελάχιστη τιμή για κάθε ℓ του κάθε υποσυνόλου $\widehat{L}$, από όλες τις ελάχιστες τιμές επιλέγεται η μέγιστη. Ο πυρήνας S που μεγιστοποιεί την τιμή της δ αποτελεί και τον πυκνότερο υπογράφο του γράφου.

Input: A multilayer graph $G = (V, E, L)$ and a real number $\beta \in \mathbb{R}^+$.
Output: The densest subgraph S^* of G.
1: $\mathbf{C} \leftarrow$ MultiLayerCoreDecomposition(G)
2: $S^* \leftarrow \arg\max_{C \in \mathbf{C}} \delta(C)$

Σχήμα 5: Ψευδοκώδικας του αλγορίθμου DensestSubgraph
(Πηγή: "Core Decomposition and Densest Subgraph in Multilayer Networks", 2017)

4.2 ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΟΥ ΣΕ ΚΩΔΙΚΑ JAVA

Για την υλοποίηση του αλγορίθμου DensestSubgraph δημιουργήθηκαν δύο συναρτήσεις που φαίνονται παρακάτω:

public static void main(String[ ] args)

Αρχικά καλείται η συνάρτηση createMultigraph( ) με όρισμα το όνομα του αρχείου από το οποίο θα διαβαστεί και θα δημιουργηθεί ο πολυεπίπεδος γράφος. Στη συνέχεια δημιουργείται ένας δισδιάστος πίνακας ακεραίων degree[ ][ ] θέσεων μήκους πλήθος επιπέδων x πλήθος κορυφών του γράφου. Δημιουργείται μια λίστα από λίστες, η cores, στην οποία εκχωρείται το αποτέλεσμα της κλήσης ενός εκ των δύο αλγορίθμων (BruteForce και BFS), δηλαδή η αποσύνθεση πυρήνα του γράφου. Ορίζονται μία λίστα από λίστες, η lList, μία λίστα που περιέχει τα επίπεδα του γράφου, η allLayers, και μία πραγματική μεταβλητή b, η τιμή της οποίας είναι στην κρίση του χρήστη. Έπειτα, κατασκευάζονται όλα τα δυνατά διακριτά υποσύνολα του συνόλου allLayers, όπου κάθε υποσύνολο βρίσκεται σε μία λίστα tempL, και προστίθενται στην λίστα lList. Δημιουργείται μία λίστα S, στην οποία θα εκχωρηθούν οι κορυφές που ανήκουν στον πυκνότερο υπογράφο, μία πραγματική μεταβλητή d και μία ακέραια μεταβλητή noe. Για κάθε πυρήνα core στη λίστα cores δημιουργούνται δύο πραγματικές μεταβλητές, max kai min με αρχικές τιμές 0 και 1000 αντίστοιχα. Για κάθε υποσύνολο- λίστα L του συνόλου επιπέδων lList, και για κάθε επίπεδο l του L, υπολογίζεται ο αριθμός των ακμών, καλώντας τη συνάρτηση numberOfEdges( ) με ορίσματα το επίπεδο l και τη λίστα core, και το αποτέλεσμα εκχωρείται στη μεταβλητή noe. Εκτελείται η πράξη $(noe * |L|\hat{} b) / |cores|$ και το αποτέλεσμα εκχωρείται σε μια πραγματική μεταβλητή result. Αν η result είναι μικρότερη της min, τότε στην min εκχωρείται η τιμή result, δηλαδή από όλα τα l του L αναζητείται η μικρότερη τιμή της result. Αν η min είναι μεγαλύτερη της max, τότε στη max εκχωρείται η τιμή της min, δηλαδή από όλα τα υποσύνολα L της lList αναζητείται η μεγαλύτερη τιμή της min. Αν η max είναι μεγαλύτερη της d, τότε στη d εκχωρείται η τιμή της max και στην λίστα S προστίθενται όλες οι κορυφές της core, δηλαδή από όλους τους πυρήνες core αναζητείται η μεγαλύτερη τιμή της max. Όταν ολοκληρωθούν όλοι οι υπολογισμοί, η λίστα S περιέχει τις κορυφές του πυρήνα με τον πυκνότερο υπογράφο του γράφου.

public static int numberOfEdges(int l, ArrayList<Integer> core)

Η συνάρτηση δέχεται σαν όρισμα ένα επίπεδο l και έναν πυρήνα, δηλαδή μία λίστα από κορυφές. Αρχικά δημιουργείται ένα αντίγραφο του γράφου, ο tempg. Στη συνέχεια, για κάθε κορυφή του γράφου tempg, αν αυτή δεν περιέχεται στη λίστα core, διαγράφεται από τον tempg. Ορίζεται μια ακέραια μεταβλητή numberOfEdges και αρχικοποιείται με μηδέν. Έπειτα, για κάθε ακμή του γράφου tempg, αν η ετικέτα (label) της ακμής είναι η τιμή του επιπέδου l, τότε η numberOfEdges αυξάνεται κατά ένα. Τέλος, επιστρέφεται ο συνολικός αριθμός των ακμών του πυρήνα στο σημείο κλήσης της συνάρτησης.

4.3 ΠΑΡΑΔΕΙΓΜΑ ΧΡΗΣΗΣ

Θα χρησιμοποιηθεί το παράδειγμα του σχήματος 2που χρησιμοποιήθηκε στην ενότητα 2.4 για την επίδειξη της λειτουργίας του αλγορίθμου DensestSubgraph. Όποιον αλγόριθμο κι αν επιλέξουμε από τους BruteForce και BFS θα έχουμε το ίδιο αποτέλεσμα αποσύνθεσης πυρήνα του γράφου. Για το συγκεκριμένο παράδειγμα επιλέχτηκε η τιμή του b = 2.2 .
Τα υπολογισμένα cores είναι: [[A, B, C, D, E, F], [A, B, D, E, F], [B, C, E, F], [A, B, D, E], [B, E, F]]
Τα επίπεδα είναι δύο, οπότε τα δυνατά υποσύνολα επιπέδων είναι τα [1, 2], [2] και [1].

Core : [A, B, C, D, E, F]
- L is: [1, 2]
  - l is: 1 result is: 6.0
  - l is: 2 result is: 5.3 ⬅
- L is: [2]
  - l is: 2 result is: 1.3
- L is: [1]
  - l is: 1 result is: 1.5

Core : [A, B, D, E, F]
- L is: [1, 2]
  - l is: 1 result is: 6.4
  - l is: 2 result is: 4.0 ⬅
- L is: [2]
  - l is: 2 result is: 1.0
- L is: [1]
  - l is: 1 result is: 1.6

Core : [B, C, E, F]
- L is: [1, 2]
  - l is: 1 result is: 4.0 ⬅
  - l is: 2 result is: 6.0
- L is: [2]
  - l is: 2 result is: 1.5
- L is: [1]
  - l is: 1 result is: 1.0

Core : [A, B, D, E]
- L is: [1, 2]
  - l is: 1 result is: 6.0
  - l is: 2 result is: 3.0 ⬅
- L is: [2]
  - l is: 2 result is: 0.75
- L is: [1]
  - l is: 1 result is: 1.5

Core : [B, E, F]
- L is: [1, 2]
  - l is: 1 result is: 4.0 ⬅
  - l is: 2 result is: 4.0
- L is: [2]
  - l is: 2 result is: 1.0
- L is: [1]
  - l is: 1 result is: 1.0

Στο παράδειγμα, η μέγιστη τιμή του result, δηλαδή η d, είναι 5.3 . Επομένως το core για το οποίο μεγιστοποιείται η d περιέχει τις κορυφές του πυκνότερου υπογράφου. Το core αυτό είναι το [A, B, C, D, E, F], δηλαδή όλες οι κορυφές του γράφου.

ΚΕΦΑΛΑΙΟ 5:
ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

ΚΕΦΑΛΑΙΟ 5: ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Τα πειράματα διεξήχθησαν σε υπολογιστή με επεξεργαστή Intel Core i7-3540M 3.00GHz και μνήμη RAM 8 GB.

Για την αξιολόγηση των αλγορίθμων υπολογισμού της αποσύνθεσης πυρήνα χρησιμοποιήθηκαν γράφοι πραγματικών δεδομένων, οι οποίοι προήλθαν από το σύνδεσμο http://deim.urv.cat/~manlio.dedomenico/data.php. Οι γράφοι που χρησιμοποιήθηκαν έχουν αριθμό ακμών μέχρι εκατοντάδες ώστε ο χρόνος εκτέλεσης να παραμένει μικρός.

Γράφοι που χρησιμοποιήθηκαν:

- **CS- AARHUS:** Ένας πολυεπίπεδος γράφος που αναπαριστά πέντε διαφορετικές σχέσεις μεταξύ υπαλλήλων του τμήματος πληροφορικής του πανεπιστημίου Aarhus (Facebook, ελεύθερος χρόνος, εργασία, συνεργατική συγγραφή, μεσημεριανό διάλειμμα).

- **PADGETT FLORENTINE FAMILIES:** Ένας γράφος δύο επιπέδων που αναπαριστά οικογένειες της Φλωρεντίας κατά την περίοδο της Αναγέννησης (συμμαχίες γάμων και επαγγελματικές σχέσεις).

- **ORYCTOLAGUS MULTIPLEX GPI NETWORK:** Ένας πολυεπίπεδος γράφος που αφορά τους homo sapiens και χρησιμοποιεί τη BioGRID, μια βάση δεδομένων γενετικής και πρωτεϊνικής αλληλεπίδρασης σε οργανισμούς.

- **KRACKHARDT HIGH TECH:** Ένας πολυεπίπεδος γράφος - κοινωνικό δίκτυο που αναπαριστά τρεις διαφορετικές σχέσεις (συμβουλή, φιλία, αναφορά σε) μεταξύ διευθυντών τμημάτων μιας εταιρείας υψηλής τεχνολογίας.

- **HEPATITUSC MULTIPLEX NETWORK:** Ένας πολυεπίπεδος γράφος που αφορά τον ιό της ηπατίτιδας c και χρησιμοποιεί τη BioGRID, μια βάση δεδομένων γενετικής και πρωτεϊνικής αλληλεπίδρασης σε οργανισμούς.

- **DANIO RERIO MULTIPLEX NETWORK:** Ένας πολυεπίπεδος γράφος που αφορά το είδος ψαριού Danio rerio c και χρησιμοποιεί τη BioGRID, μια βάση δεδομένων γενετικής και πρωτεϊνικής αλληλεπίδρασης σε οργανισμούς.

Στον πίνακα που ακολουθεί φαίνεται για κάθε γράφο από αυτούς που περιγράφηκαν, ο αριθμός επιπέδων, ο αριθμός κορυφών και ο αριθμός ακμών του.

Graph	Layers	Vertices	Edges
AARHUS	5	61	620
PADGETT	2	16	35
ORYCTOLAGUS	3	144	144
KRACKHARDT	3	21	312
HEPATITUSC	3	105	138
DANIO RERIO	5	155	183

Σχήμα 6: Πίνακας χαρακτηριστικών των γράφων

Μετά την εκτέλεση των δύο αλγορίθμων για τους γράφους του πίνακα, προέκυψαν τα εξής αποτελέσματα σχετικά με τον αριθμό των πυρήνων του γράφου, το χρόνο εκτέλεσης σε msec και το πλήθος των υπολογισμένων πυρήνων για καθένα από τους δύο αλγορίθμους BFS και BruteForce:

Graph	Cores	BFS Cores	BFS Time	BruteForce Cores	BruteForce Time
PADGETT	8	11	165	48	180
HEPATITUSC	10	19	466	681472	-
ORYCTOLAGUS	10	17	501	49303	-
KRACKHARDT	21	93	682	323232	630615
DANIO RERIO	19	52	1483	371293	-
AARHUS	217	521	4225	17210368	-

Σχήμα 7: Πίνακας συγκριτικών αποτελεσμάτων BFS και BruteForce

Λόγω του μεγάλου αριθμού διανυσμάτων k, ο οποίος προκύπτει από τον αριθμό των επιπέδων επί το μέγιστο βαθμό του γράφου (οπότε όσο περισσότερες ακμές τόσο πιο μεγάλος ο βαθμός κορυφής), ο BruteForce έχει χειρότερη απόδοση υπολογίζοντας όλους τους πυρήνες των διανυσμάτων k. Στον πίνακα με παύλα σημειώνονται οι χρόνοι που ξεπέρασαν το 1.5M msec (περίπου 30'). Ενδιαφέρον συγκρίνοντας τους δύο αλγορίθμους έχει η τεράστια διαφορά στον χρόνο εκτέλεσης μεταξύ των δύο αλγορίθμων, αλλά και το πλήθος των πυρήνων που υπολογίζονται.

Στη συνέχεια, πραγματοποιήθηκαν πειράματα για τον αλγόριθμο DensestSubgraph, στους ίδιους γράφους, εναλλάσσοντας τις τιμές της μεταβλητής b. Τα αποτελέσματα συγκλίνουν στο ότι όσο πιο μεγάλη τιμή έχει το b, τόσο λιγότερες κορυφές περιέχει ο πυκνότερος υπογράφος και τόσο μεγαλύτερη είναι η τιμή της δ (πυκνότητας).

ΚΕΦΑΛΑΙΟ 6:
ΣΥΜΠΕΡΑΣΜΑΤΑ

ΚΕΦΑΛΑΙΟ 6: ΣΥΜΠΕΡΑΣΜΑΤΑ

Στόχος της παρούσας εργασίας είναι η παρουσίαση μεθόδων υπολογισμού της αποσύνθεσης πυρήνα σε πολυεπίπεδους γράφους και εξαγωγής του πυκνότερου υπογράφου από πολυεπίπεδο γράφο.

Εξετάστηκαν δύο αλγόριθμοι για τον υπολογισμό της αποσύνθεσης πυρήνα. Ο πρώτος ήταν ο BruteForce, αλγόριθμος ωμής βίας, και ο δεύτερος ήταν ο BFS. Ο BruteForce υπολόγιζε τους πυρήνες ενός γράφου για όλα τα δυνατά διανύσματα k χωρίς κάποιο περιορισμό του χώρου αναζήτησης, ενώ ο BFS ακολουθώντας κάποιους αποτελεσματικούς κανόνες κλαδέματος του πλέγματος πέτυχε μείωση του πλήθους υπολογισμών πυρήνων, αλλά και καλύτερους χρόνους εκτέλεσης.

Όπως ήταν αναμενόμενο, ο αλγόριθμος BFS είχε καλύτερους χρόνους εκτέλεσης από τον BruteForce για τα πειράματα που πραγματοποιήθηκαν, λόγω του ότι ο χρόνος εκτέλεσης του BruteForce αυξάνεται εκθετικά με τον αριθμό των ακμών.

Επίσης, εξετάστηκε ο αλγόριθμος DensestSubgraph για την εξαγωγή του πυκνότερου υπογράφου. Παρατηρήθηκε ότι για μεγάλες τιμές του b προέκυπτε μεγαλύτερη πυκνότητα στον υπογράφο, με μικρότερο σύνολο κορυφών.

Όπως ήδη έχει αναφερθεί, η υλοποίηση του κώδικα των αλγορίθμων εκπονήθηκε σε γλώσσα Java. Ένα πρώτο εμπόδιο που εμφανίστηκε κατά την υλοποίηση ήταν ότι, ενώ η βιβλιοθήκη JGraphT παρέχει δομές και συναρτήσεις για γράφους, δεν υπήρχε ολοκληρωμένη δομή για κατασκευή πολυεπίπεδων γράφων, οπότε χρησιμοποιήθηκε η δομή MultilayerGraph στη γενική της μορφή, με την διαφορά ότι οι ακμές του γράφου δεν δημιουργούνταν από την κλάση DefaultEdge, αλλά δημιουργήθηκε μία νέα ώστε να δέχεται το όρισμα του επιπέδου ως ετικέτα στις αντίστοιχες ακμές.

Ένα ακόμη εμπόδιο ήταν η επιλογή των διάφορων δομών που χρησιμοποιήθηκαν σε διάφορα σημεία των αλγορίθμων. Σε μία άλλη γλώσσα, όπως η Python, για την οποία έχουν αναπτυχθεί πολλές βιβλιοθήκες, η κατασκευή των δομών που απαιτούνται από τους αλγορίθμους ίσως ήταν πιο εύκολη και ενδεχομένως οι αλγόριθμοι να εκτελούνταν σε λιγότερο χρόνο.

Τα ανοιχτά ζητήματα που θα μπορούσαν να διερευνηθούν από μία μελλοντική επέκταση της παρούσας εργασίας είναι η μελέτη δύο ακόμα αλγορίθμων που παρουσιάζονται στο έργο με τίτλο “Core Decomposition and Densest Subgraph in Multilayer Networks”, στο οποίο και βασίστηκε η παρούσα εργασία, ο DFS και ο Hybrid. Επίσης προτείνεται η χρήση της γλώσσας Python λόγω των μεγάλων προγραμματιστικών δυνατοτήτων της. Επιπρόσθετα, προτείνεται η παράλληλη επεξεργασία για αύξηση των υπολογιστικών επιδόσεων.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Edoardo Galimberti, Francesco Bonchi, Francesco Gullo. Core Decomposition and Densest Subgraph in Multilayer Networks. 2017

[2] Fragkiskos D. Malliaros, Apostolos N. Papadopoulos, Michalis Vazirgiannis. Core Decomposition in Graphs: Concepts, Algorithms and Applications.

[3] V. Batagelj, M. Zaversnik. An O(m) Algorithm for Cores Decomposition of Networks. 2003

[4] J. Cheng, Y. Ke, S. Chu, and M. T. Ozsu. Efficient core decomposition in massive networks. 2011

[5] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and U. V. Çatalyürek. Streaming algorithms for k-core decomposition. Proc. VLDB Endow., Apr. 2013.

[6] Ahmet Erdem Sarıyüce, Bugra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, Ümit V. Çatalyürek. Incremental k-core decomposition: algorithms and evaluation. 2014