ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αλγόριθμοι σε Spark για την εύρεση κοινοτήτων σε πολυεπίπεδα σύνθετα δίκτυα

Spark-based algorithms for finding communities in multi-layer complex networks

Διπλωματική Εργασία
υπό
Κόιου Λάζαρο

Επιβλέποντες:	Κατσαρός Δημήτριος	Μποζάνης Παναγιώτης
	Επίκουρος Καθηγητής Π.Θ.	Καθηγητής Π.Θ

Βόλος, 2017

Αυτή η σελίδα είναι σκόπιμα λευκή

Διπλωματική εργασία για την απόκτηση του διπλώματος του Μηχανικού Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του προγράμματος προπτυχιακών σπουδών του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών της Πολυτεχνικής Σχολής του Πανεπιστημίου Θεσσαλίας.

……
Λάζαρος Κόιου

Διπλωματούχος Μηχανικός Ηλεκτρονικών Υπολογιστών, Τηλεπικοινωνιών και Δικτύων Πανεπιστημίου Θεσσαλίας

Αυτή η σελίδα είναι σκόπιμα λευκή

Ευχαριστίες

Με την περάτωση της διπλωματικής εργασίας θα ήθελα να ευχαριστήσω έναν μεγάλο αριθμό ατόμων, χωρίς να αναφερθώ με ονόματα. Θα ήθελα να ευχαριστήσω όλους όσους με βοήθησαν κατά τη διάρκεια των σπουδών μου, άσχετα με το πόσο.

Επίσης, ευχαριστώ θερμά τους επιβλέποντες της διπλωματικής εργασίας κ. Παναγιώτη Μποζάνη και ιδιαίτερα τον κ. Δημήτριο Κατσαρό για όλη την βοήθεια και την αμεσότητα κατά τη διάρκεια της εργασίας, από την επιλογή του θέματος μέχρι το τελείωμα. Ακόμα, θα ήθελα να ευχαριστήσω τον διδακτορικό φοιτητή Παύλο Μπασάρα για την παροχή δικτύων κατά τη διάρκεια των πειραμάτων.

Τέλος, θα ευχαριστήσω την οικογένεια μου για τη συνεχή στήριξη και βοήθεια όλα αυτά τα χρόνια και όσους ήταν δίπλα μου.

Λάζαρος Κόιου
Βόλος, 2017

Αυτή η σελίδα είναι σκόπιμα λευκή

Περίληψη

Η εύρεση στενά συνδεδεμένων υπογραφημάτων, όπως είναι οι κλίκες ή οι quasi-cliques, είναι ένα σημαντικό πρόβλημα στην επιστήμη των σύνθετων δικτύων. Σημαντικές εφαρμογές βρίσκονται στα κοινωνικά δίκτυα και στην βιολογία, μεταξύ άλλων, όπου οι πληροφορίες μπορούν να αναπαρασταθούν ως γραφήματα, καθένα από τα οποία αναπαριστά κάποιου είδους σχέσης μεταξύ των οντοτήτων. Η συλλογή όλων αυτών των γραφημάτων σε ένα μοναδικό γράφημα δημιουργεί ένα πολύ-επίπεδο γράφημα. Χρησιμοποιώντας την πληροφορία από κάθε επίπεδο του γραφήματος μπορούμε να ανακαλύψουμε τις πιο ενδιαφέρουσες ομάδες οντοτήτων.

Η μελέτη, ο σχεδιασμός και η ανάπτυξη λογισμικού εύρεσης κοινοτήτων είναι το αντικείμενο της παρούσας διπλωματικής εργασίας. Βασισμένο στον αλγόριθμο MiMAG (Mining Multi-layered, Attributed Graphs) αναπτύχθηκε το λογισμικό MCD (Multi-layer Community Detector) στις γλώσσες Scala και Spark. Ο MCD δέχεται ως είσοδο ένα αρχείο το οποίο περιέχει ένα πολυεπίπεδο δίκτυο και εντοπίζει τα πιο στενά και ενδιαφέροντα υπογραφήματα.

Στο κεφάλαιο 1 γίνεται η παρουσίαση των γλωσσών και των βιβλιοθηκών που χρησιμοποιήθηκαν και εξηγείται ο τρόπος εγκατάστασης τους. Το κεφάλαιο 2 περιέχει τις εισαγωγικές έννοιες. Το κεφάλαιο 3 περιέχει μία ανασκόπηση στη θεωρία. Στο κεφάλαιο 4 παρουσιάζεται και αναλύεται το μοντέλο MLCS, το μοντέλο των cluster, το μοντέλο ομαδοποίησης τους και η πλεονάζουσα σχέση. Στο κεφάλαιο 5 αναλύεται ο αλγόριθμος MiMAG και η ποιότητα των υποδέντρων. Το κεφάλαιο 6 περιέχει τις διάφορες τεχνικές κλαδέματος και βελτίωσης του λογισμικού. Στο κεφάλαιο 7 εξηγείται το προγραμματιστικό μέρος του λογισμικού και παρουσιάζονται τα μηνύματα που εμφανίζονται κατά την εκτέλεση του. Το κεφάλαιο 8 είναι το πειραματικό μέρος της εργασίας και περιέχει την παρουσίαση των δικτύων στα οποία διεξάχθηκαν τα πειράματα και τα αποτελέσματα. Τέλος, στο κεφάλαιο 9 βρίσκεται η σύνοψη των δυνατοτήτων και αδυναμιών του λογισμικού και αναφορά σε πιθανές μελλοντικές επεκτάσεις.

Abstract

Finding closely connected subgraphs, such as cliques or quasi-cliques, is an important problem in the science of complex networks. Many applications can be found in social networks and biology, among others, where information can be depicted as graphs, where each graph represents some type of relation between the existing entities. The collection of all of these graphs in a single one forms a multi-layer graph. Using the information of each layer of the graph we can discover the most interesting group of entities among them.

The study, the design and the development of a software that can find communities is the objective of the current thesis. Based on the MiMAG algorithm (Mining Multi-layered, Attributed Graphs) the software MCD (Multi-layer Community Detector) was developed in the languages Scala and Spark. MCD takes as input a file that contains a multi-layer network and finds the closely connected and interesting subgraphs.

In chapter 1 is the presentation of the languages and libraries that have been used in this project and the way of their installation. Chapter 2 contains introductory meanings. Chapter 3 contains a review of background theory. In chapter 4 the MLCS model, the cluster's model, the clustering's model, the quality and the redundant relation are all presented and analyzed. In chapter 5 the MiMAG algorithm and the quality for subtrees are analyzed. Chapter 6 contains various pruning techniques and improvements of the software. In chapter 7 the software's development is explained and the output messages during execution are presented. Chapter 8 is the experimental part of this project and contains the networks in which the experiments were conducted and the results. Finally, chapter 9 contains the summary of the software's features and weaknesses and reference to possible feature enhancements.

Περιεχόμενα

Ευχαριστίες v
Περίληψη vii
Abstract viii
Περιεχόμενα ix
Πίνακας Εικόνων x
1. Εισαγωγή 1
1.1 Λογισμικά που χρησιμοποιήθηκαν (Scala) 1
1.2 Λογισμικά που χρησιμοποιήθηκαν (Spark) 2
1.3 Λογισμικά που χρησιμοποιήθηκαν (IntelliJ) 3
1.4 Στήσιμο του Spark 2.0 με Scala API σε Windows 10 στο IntelliJ έκδοσης 2016.3.3 και δημιουργία ενός project 4
2. Εισαγωγή 10
2.1 Εισαγωγικά για τα Σύνθετα Δίκτυα 10
2.2 Γνωστικό υπόβαθρο 14
3. Ανασκόπηση 16
3.1 Είδη Αλγορίθμων 16
3.2 Προγενέστεροι Αλγόριθμοι 17
4. Μοντελοποίηση 19
4.1 Σύμβολα που θα χρησιμοποιηθούν 19
4.2 Μοντέλο MLCS 20
4.3 Μοντέλο των cluster 21
4.4 Ποιότητα ενός cluster 25
4.5 Μοντέλο ομαδοποίησης κορυφών 26
4.6 Πλεονάζουσα σχέση ανάμεσα σε cluster 27
5. MiMAG 29
5.1 Ο αλγόριθμος MiMAG 29
5.2 Εκτέλεση του αλγόριθμου MiMAG 33
5.3 Ποιότητα υποδέντρων 35
6. Τεχνικές κλαδέματος και βελτίωσης 37
6.1 Τεχνικές κλαδέματος 37
6.2 Τεχνικές βελτίωσης 39
7. Προγραμματιστικό μέρος - Έξοδος προγράμματος 40
7.1 Προγραμματιστικό μέρος 40
7.2 Έξοδος προγράμματος 42
8. Πειραματικό μέρος 46
8.1 Περιγραφή συστήματος και προδιαγραφών 46
8.2 1ο Δίκτυο 47
8.3 2ο Δίκτυο 50
8.4 3ο Δίκτυο 53
8.5 4ο Δίκτυο 55
8.6 Περαιτέρω δίκτυα και αποτελέσματα 58
9. Σύνοψη – Αδυναμίες λογισμικού / Μελλοντικές επεκτάσεις 59
9.1 Σύνοψη 59
9.2 Αδυναμίες λογισμικού και μελλοντικές επεκτάσεις 60
10. Παράρτημα 61
11. Βιβλιογραφία 77

Πίνακας Εικόνων

Εικόνα 1 - Πρόγονοι της Scala 1
Εικόνα 2 – Το οικοσύστημα της Spark 2
Εικόνα 3 – Το περιβάλλον ανάπτυξης λογισμικού IntelliJ 3
Εικόνα 4 – New Environment Variable 4
Εικόνα 5 - Create New Project Screen 5
Εικόνα 6 - Εγκατάσταση Scala plugin, SBT plugin 5
Εικόνα 7 - Δημιουργία Scala SBT Project 6
Εικόνα 8 - Project Configurations 6
Εικόνα 9 - Waiting for Index Completion 7
Εικόνα 10 - Πρόσθεση Βιβλιοθηκών 7
Εικόνα 11 - Βιβλιοθήκες 7
Εικόνα 12 - Επιλογή των modules 8
Εικόνα 13 - Δοκιμή 1 8
Εικόνα 14 - Δοκιμή 2 9
Εικόνα 15 - Δομική 3, Κώδικας 9
Εικόνα 16 - Δοκιμή 4, Output 9
Εικόνα 17 – Γράφημα ενός δικτύου 10
Εικόνα 18 – Το γράφημα ενός σύνθετου δικτύου 11
Εικόνα 19 - Το πρόβλημα των 7 γεφυρών 12
Εικόνα 20 - Παραδείγματα χρήσης των σύνθετων δικτύων 13
Εικόνα 21 - Απλό, μη κατευθυνόμενο γράφημα 14
Εικόνα 22 – Πολυδιάστατο γράφημα 4 διαστάσεων 14
Εικόνα 23 - Γράφημα χωρισμένο σε κοινότητες 15
Εικόνα 24 - Παράδειγμα υπογραφήματος 21
Εικόνα 25 - Υπογραφήματα παραδείγματος 22
Εικόνα 26 - Υπογράφηματα παραδείγματος 24
Εικόνα 27 - Υπογράφημα παραδείγματος 26
Εικόνα 28 - Cluster παραδείγματος 27
Εικόνα 29 - Set enumeration tree 29
Εικόνα 30 – Extended set enumeration tree με 3 κορυφές και 2 layer 31
Εικόνα 31 - Αλγόριθμος MiMAG 32
Εικόνα 32 - Σχήμα επέκτασης ενός κόμβου 33
Εικόνα 33 - Παράδειγμα διαγραφής υποδέντρου που ανήκει σε 1 layer 38
Εικόνα 34 - Κλάση node 40
Εικόνα 35 - Έξοδος 1 42
Εικόνα 36 - Έξοδος 2 42
Εικόνα 37 - Έξοδος 3 42
Εικόνα 38 - Έξοδος 4 43
Εικόνα 39 - Έξοδος 5 43
Εικόνα 40 - Έξοδος 6 43

Εικόνα 41 - Έξοδος 7....43
Εικόνα 42 - Έξοδος 8....44
Εικόνα 43 - Έξοδος 9....44
Εικόνα 44 - Έξοδος 10....44
Εικόνα 45 - Έξοδος 11....44
Εικόνα 46 - Έξοδος 12....45
Εικόνα 47 - Έξοδος 13....45
Εικόνα 48 - Απεικόνιση του δικτύου Arabidopsis....48
Εικόνα 49 - Παραδειγμα αποτελέσματος 1....49
Εικόνα 50 - Απεικόνιση του δικτύου Auger....51
Εικόνα 51 - Παραδειγμα αποτελέσματος 2....52
Εικόνα 52 - Απεικόνιση του δικτύου C.Elegans....53
Εικόνα 53 - Παραδειγμα αποτελέσματος 3....54
Εικόνα 54 - Απεικόνιση του δικτύου Sacchpomb....56
Εικόνα 55 - Παράδειγμα αποτελέσματος 4....57
Πίνακας 1 - Συνοπτικά αποτελέσματα πειραμάτων....58

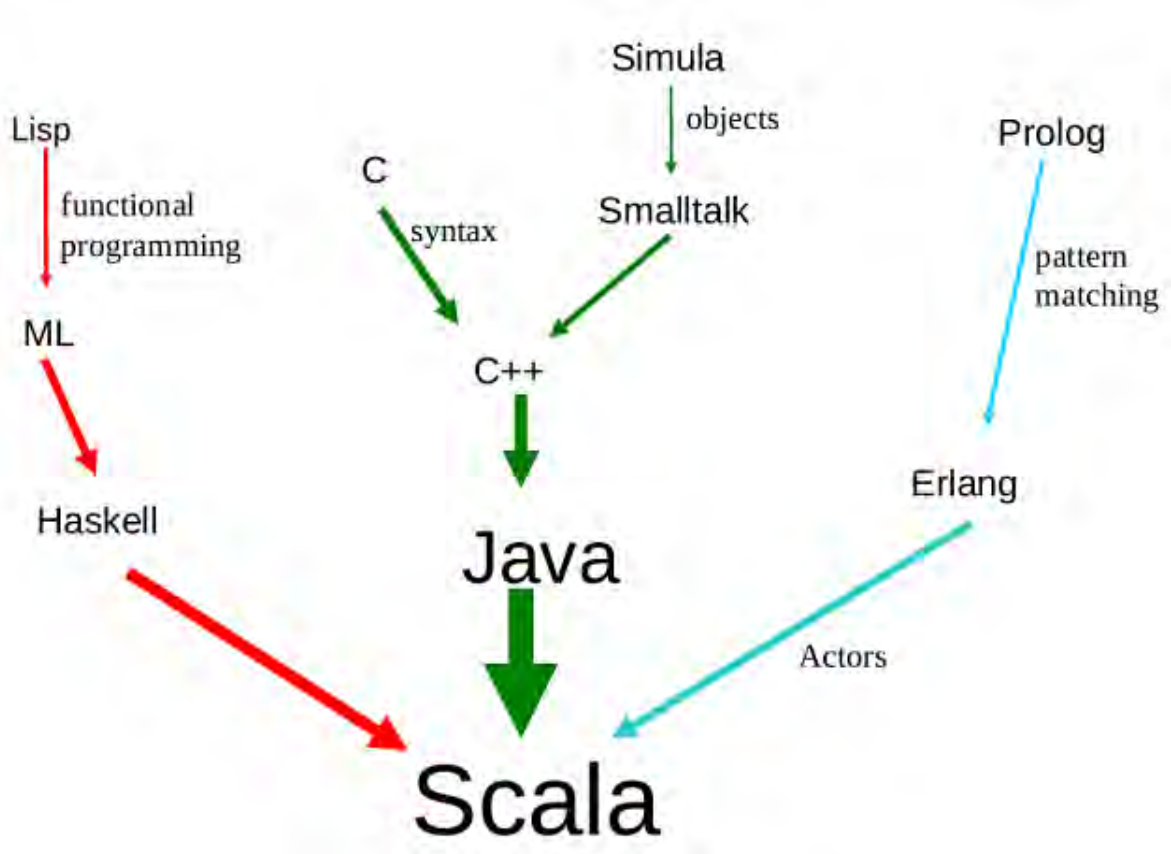
1. Εισαγωγή

1.1 Λογισμικά που χρησιμοποιήθηκαν (Scala)

Η γλώσσα προγραμματισμού Scala που χρησιμοποιήθηκε για την ανάπτυξη του προγράμματος είναι ακρωνύμιο για “Scalable Language”. Η πρώτη της έκδοση ήταν τον Ιανουάριο του 2004 και η έκδοση που χρησιμοποιήθηκε στο πρόγραμμα είναι η 2.11.8 .

Η Scala υποστηρίζει το αντικειμενοστραφή και το συναρτησιακό μοντέλο προγραμματισμού και έχει υιοθετήσει ένα μεγάλο μέρος του συντακτικού της από την Java και C# .

Εικόνα 1 - Πρόγονοι της Scala



Πηγή: Παπαδόπουλος, 2015

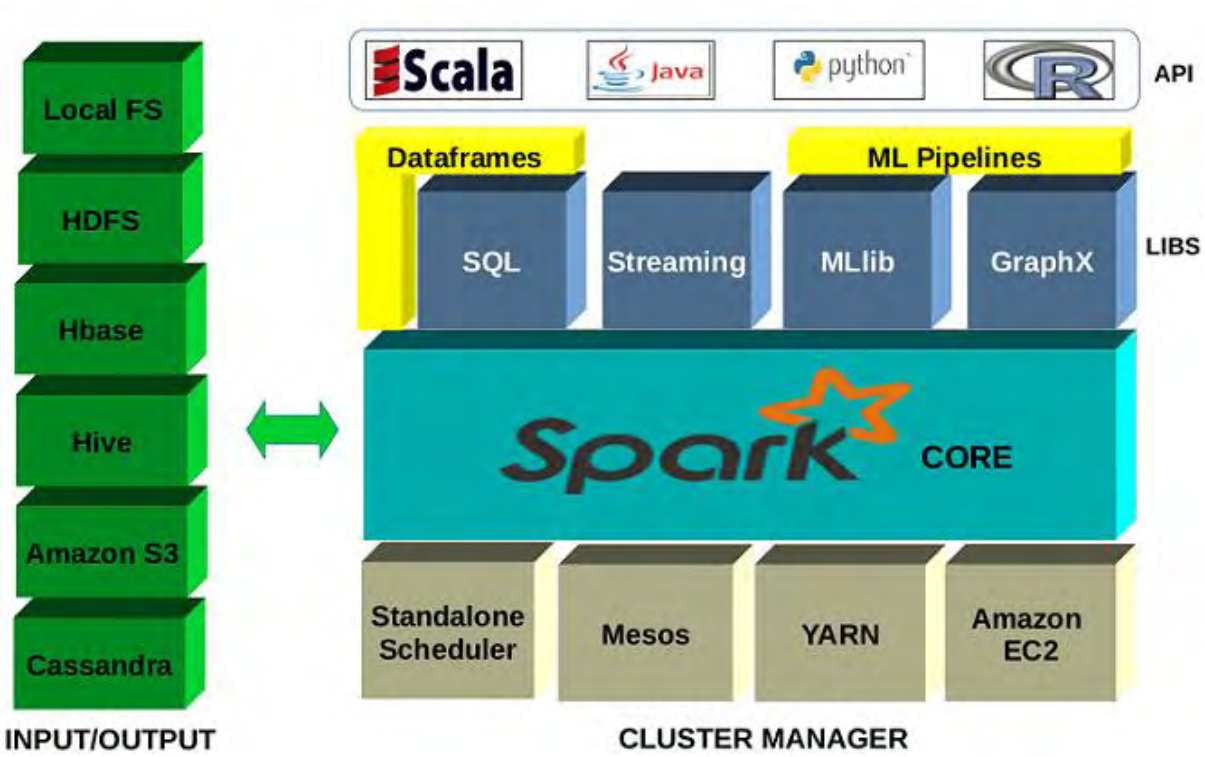
1.2 Λογισμικά που χρησιμοποιήθηκαν (Spark)

Η Spark είναι μια ενοποιημένη πλατφόρμα για cluster computing, που καθιστά ικανή την αποτελεσματική διαχείριση, ανάλυση και επεξεργασία για big data. Έχει API για Scala, Java, Python, R. Είναι συμβατή με δεδομένα σε μορφή HDFS, HBase, Cassandra, Hive, και κάθε Hadoop InputFormat.

Οι βιβλιοθήκες που υπάρχουν αυτή τη στιγμή είναι:
- SQL Lib
- Streaming Lib
- Machine Learning Lib (MLlib)
- Graph Lib (GraphX)

Στο πρόγραμμα χρησιμοποιήθηκε εκτενώς το GraphX το οποίο περιέχει API για επεξεργασία γραφημάτων. Επίσης, περιέχει υλοποιημένους αλγόριθμους όπως PageRank, Connected components, Label propagation, SVD++, Strongly connected components, Triangle counting, Core decomposition, και άλλους, που δεν χρησιμοποιήθηκαν.

Εικόνα 2 – Το οικοσύστημα της Spark

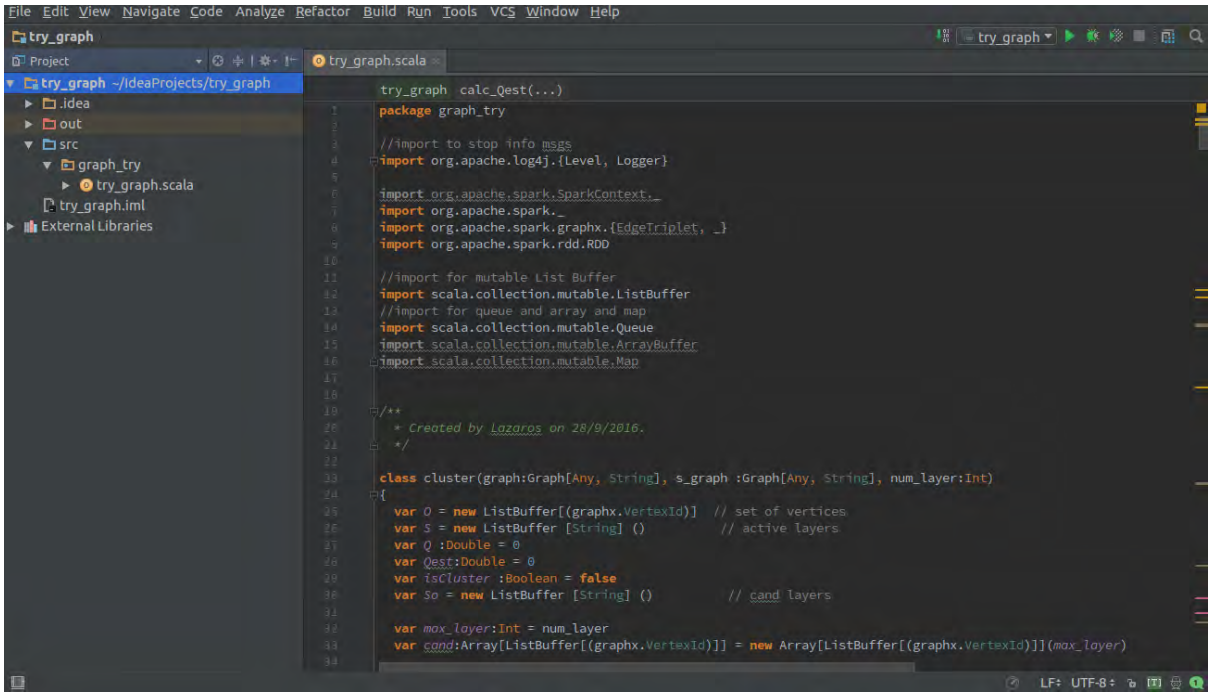


Πηγή: Παπαδόπουλος, 2015

1.3 Λογισμικά που χρησιμοποιήθηκαν (IntelliJ)

Για την συγγραφή του κώδικα χρησιμοποιήθηκε το πρόγραμμα IntelliJ. Το IntelliJ είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού και μπορεί να τρέξει σε Windows, Mac OS και Linux. Μπορεί να υποστηρίξει Scala και μπορεί κανείς εύκολα να προσθέσει τις βιβλιοθήκες για το Spark.

Εικόνα 3 – Το περιβάλλον ανάπτυξης λογισμικού IntelliJ



Πηγή: Ιδία επεξεργασία

1.4 Στήσιμο του Spark 2.0 με Scala API σε Windows 10 στο IntelliJ έκδοσης 2016.3.3 και δημιουργία ενός project.

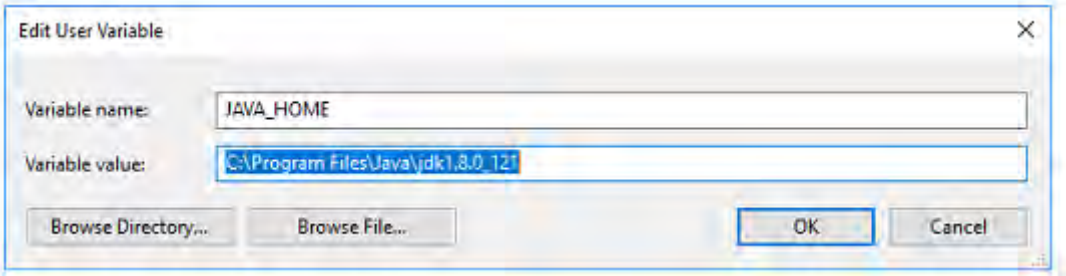
1. Αρχικά χρειάζεται να κατεβάσουμε και να εγκαταστήσουμε το πρόγραμμα ανάπτυξης λογισμικού IntelliJ. Η παρούσα έκδοση είναι η 2016.3.3 και μπορεί να βρεθεί στον παρακάτω σύνδεσμο:
   https://www.jetbrains.com/idea/download/#section=windows

2. Έπειτα, πρέπει να κατεβάσουμε την τελευταία έκδοση του Oracle/Sun JDK.
   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

3. Πρέπει να δημιουργήσουμε μια Environment Variable. Αυτό μπορεί να γίνει πηγαίνοντας:

   My Computer → Properties → Advanced system settings → Environment Variables → New

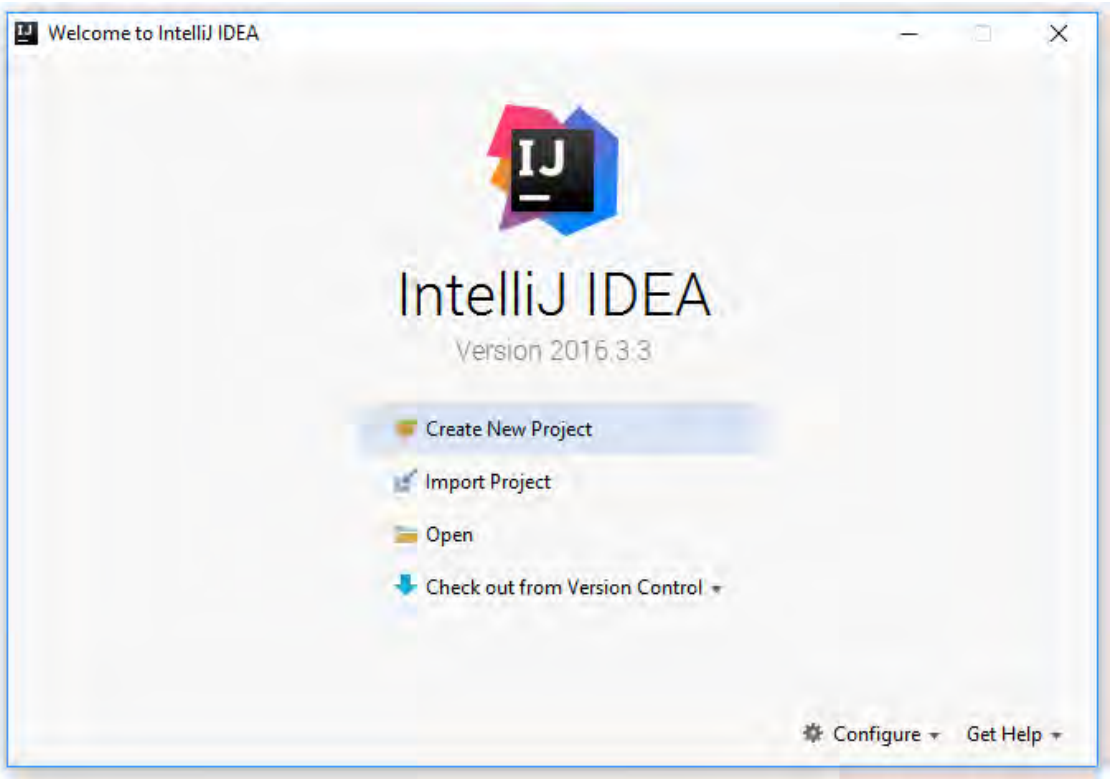
Εικόνα 4 – New Environment Variable



Πηγή: Ιδία επεξεργασία

4. Ανοίγοντας το IntelliJ παραλείπουμε τις διαμορφώσεις και δεν χρειάζεται να εγκαταστήσουμε τίποτα, μέχρι να βρεθούμε στο σημείο ‘create new project’:

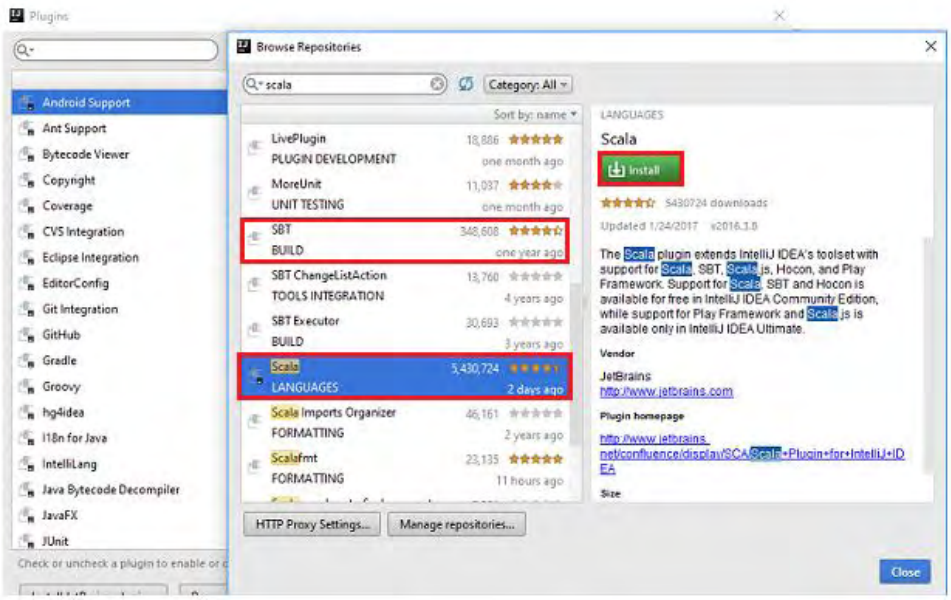
Εικόνα 5 - Create New Project Screen



Πηγή: Ιδία επεξεργασία

5. Στο κάτω μέρος της εικόνας πηγαίνουμε στο Configure → Plugins → Browse repositories και εγκαθιστούμε το Scala plugin και το SBT plugin.

Εικόνα 6 - Εγκατάσταση Scala plugin, SBT plugin

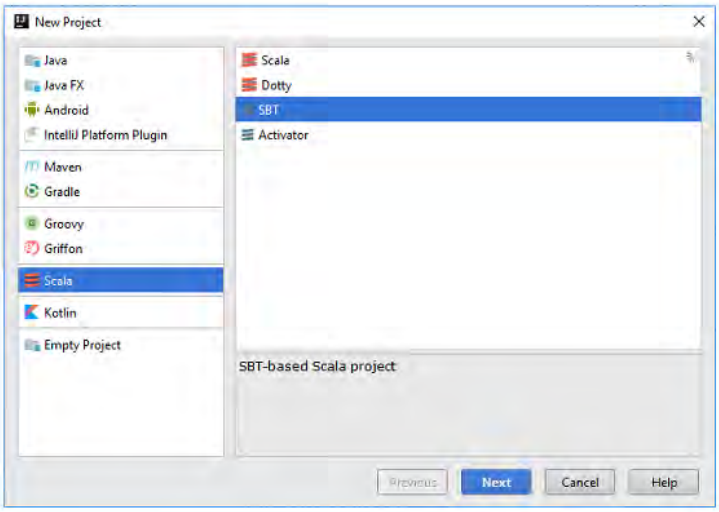


Πηγή: Ιδία επεξεργασία

6. Κάνουμε επανεκκίνηση το IntelliJ όπως ζητείται.

7. Δημιουργούμε ένα καινούργιο Scala SBT Project:

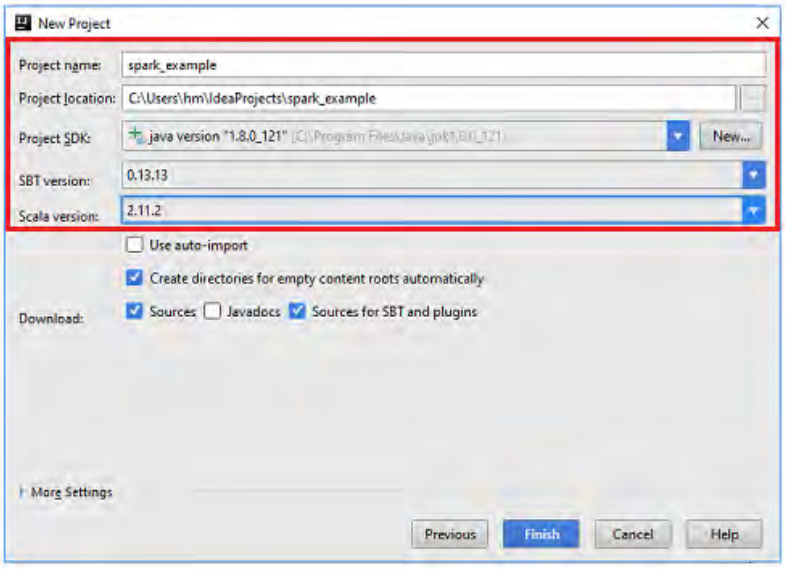
Εικόνα 7 - Δημιουργία Scala SBT Project



Πηγή: Ιδία επεξεργασία

8. Δημιουργούμε το project με τις ακόλουθες ρυθμίσεις. Προσοχή χρειάζεται στην επιλογή της έκδοσης της Scala, γιατί πρέπει να υποστηρίζεται από την έκδοση του Spark.

Εικόνα 8 - Project Configurations



Πηγή: Ιδία επεξεργασία

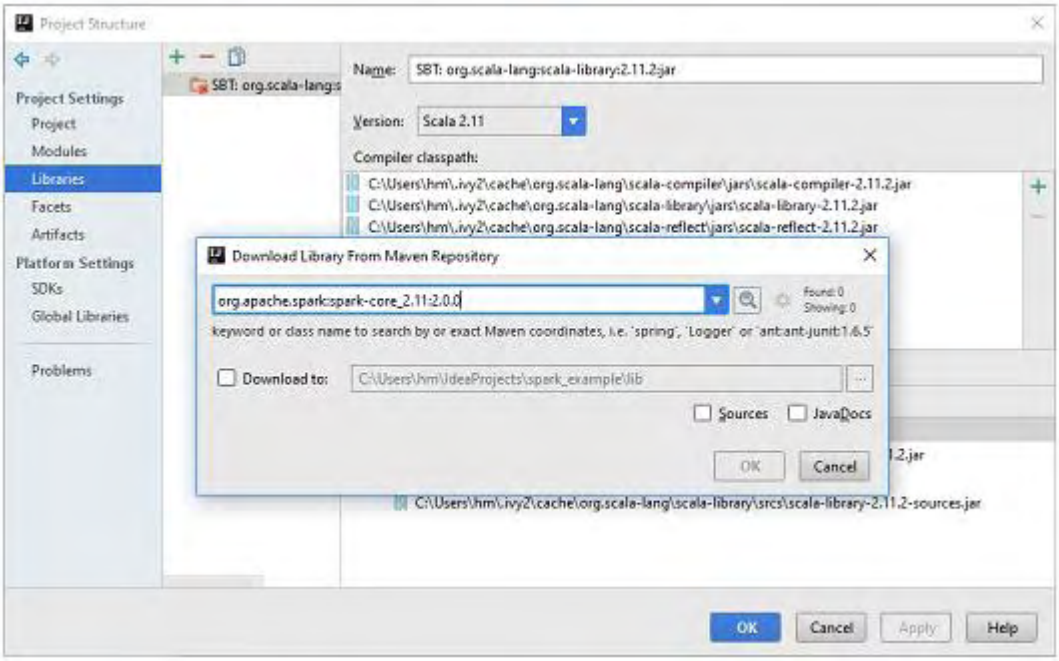
9. Περιμένουμε μέχρι να ολοκληρωθούν οι διεργασίες αρχικοποίησης του IntelliJ!

Εικόνα 9 - Waiting for Index Completion

Πηγή: Ιδία επεξεργασία

10. Σε αυτό το σημείο χρειάζεται να φτιάξουμε τις ρυθμίσεις για τις βιβλιοθήκες για το νέο project. Πηγαίνουμε στο File → Project Structure → Libraries και προσθέτουμε τις βιβλιοθήκες από το Maven όπως φαίνεται παρακάτω:

Εικόνα 10 - Πρόσθεση Βιβλιοθηκών



Πηγή: Ιδία επεξεργασία

Οι βιβλιοθήκες που πρέπει να προστεθούν είναι:

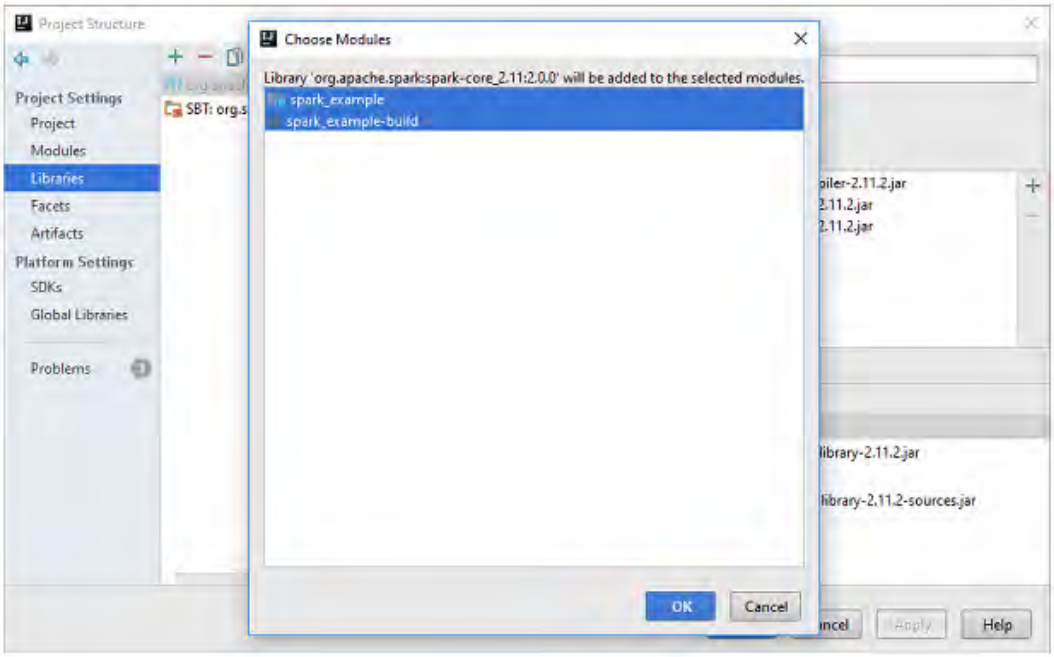
Εικόνα 11 - Βιβλιοθήκες

Package	Artifact ID
org.apache.spark:spark-core_2.11:2.0.0	spark-core_2.11
org.apache.spark:spark-sql_2.11:2.0.0	spark-sql_2.11
org.apache.spark:spark-mllib_2.11:2.0.0	spark-mllib_2.11
org.apache.spark:spark-streaming_2.11:2.0.0	spark-streaming_2.11
org.apache.spark:spark-graphx_2.11:2.0.0	spark-graphx_2.11

Πηγή: Nahoom-Kabakov, 2016

Καθώς προσθέτουμε την κάθε μία βιβλιοθήκη πρέπει να επιλέξουμε τα εξής modules:

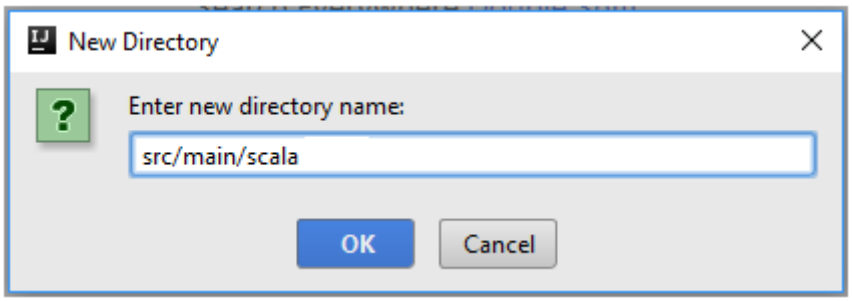
Εικόνα 12 - Επιλογή των modules



Πηγή: Ιδία επεξεργασία

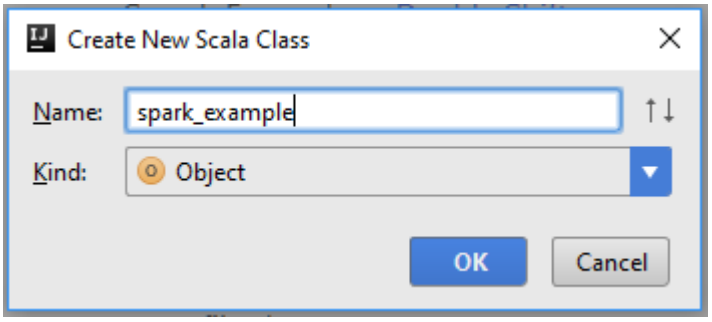
11. Μόλις τελειώσει το IntelliJ την επεξεργασία των ρυθμίσεων μπορούμε να δοκιμάσουμε αν δουλεύουν. Πηγαίνουμε στο project μας spark_example → New Directory:

Εικόνα 13 - Δοκιμή 1



Πηγή: Ιδία επεξεργασία

Εικόνα 14 - Δοκιμή 2



Πηγή: Ιδία επεξεργασία

Ο δοκιμαστικός κώδικας μπορεί να είναι ο εξής:

Εικόνα 15 - Δομική 3, Κώδικας

```
1   import org.apache.spark.{SparkConf, SparkContext}
2
3   /**
4     * Created by Lazaros.
5     */
6   object spark_example {
7
8     def main(args: Array[String]): Unit = {
9
10      val conf = new SparkConf()
11        .setAppName("Load graph")
12        .setMaster("local")
13
14      val sc = new SparkContext(conf)
15      println(sc)
16
17    }
18  }
```

Πηγή: Ιδία επεξεργασία

Το σωστό output μοιάζει κάπως έτσι:

Εικόνα 16 - Δοκιμή 4, Output

Πηγή: Ιδία επεξεργασία

2. Εισαγωγή

2.1 Εισαγωγικά για τα Σύνθετα Δίκτυα

Δίκτυο θεωρείται μια συλλογή οντοτήτων που είναι διασυνδεδεμένες με κάποιους δεσμούς και μπορεί να αναπαρασταθεί με ένα γράφημα. Η οντότητα μπορεί να είναι οτιδήποτε (ένας άνθρωπος, ένας οργανισμός, ένας υπολογιστής, κ.α.) και η σύνδεση τους υποδηλώνει μια σχέση μεταξύ των κόμβων (φιλία, συνεργασία, επαφή, κ.α.).

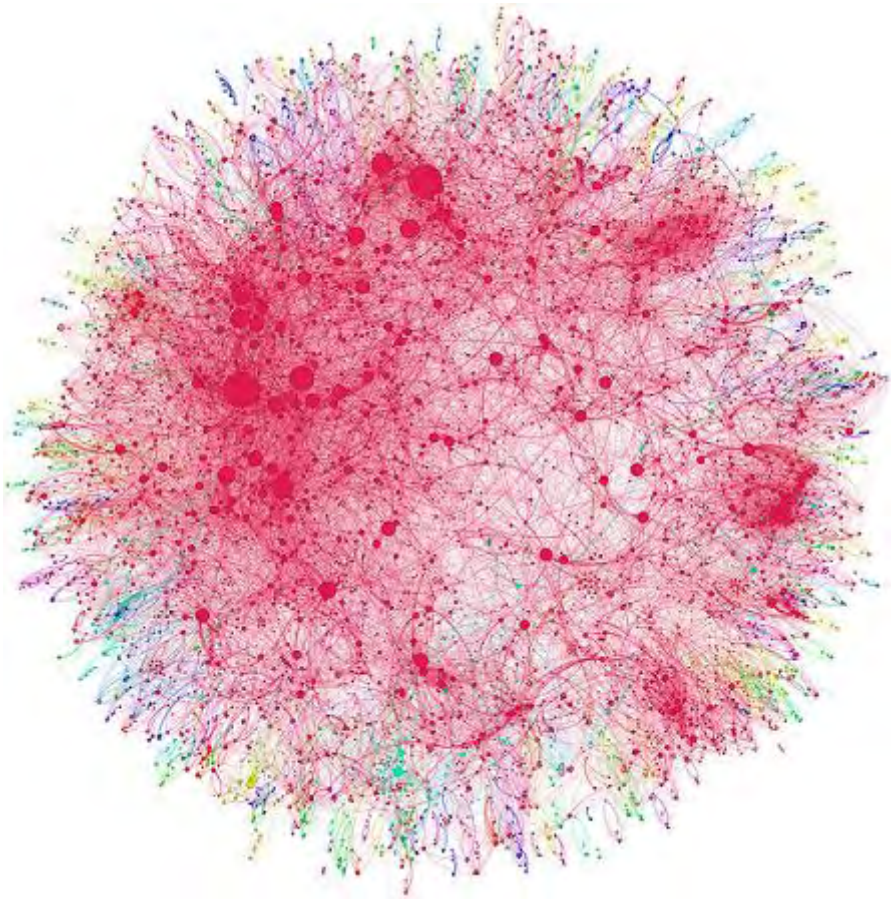
Εικόνα 17 – Γράφημα ενός δικτύου

Πηγή: http://www.recruitingunblog.com/

Ένα σύνθετο δίκτυο είναι ένα δίκτυο με μεγάλο αριθμό διασυνδεδεμένων κόμβων. Αυτό που καθιστά τα δίκτυα σύνθετα είναι ότι είναι τόσο μεγάλα ώστε να είναι αδύνατον να καταλάβουμε ή να προβλέψουμε την συμπεριφορά τους κοιτώντας την συμπεριφορά

μεμονωμένων κόμβων ή συνδέσεων. Η επιστήμη των σύνθετων δικτύων επιδιώκει να περιγράψει τις ιδιότητες και τα χαρακτηριστικά αυτών των δικτύων.

Εικόνα 18 – Το γράφημα ενός σύνθετου δικτύου

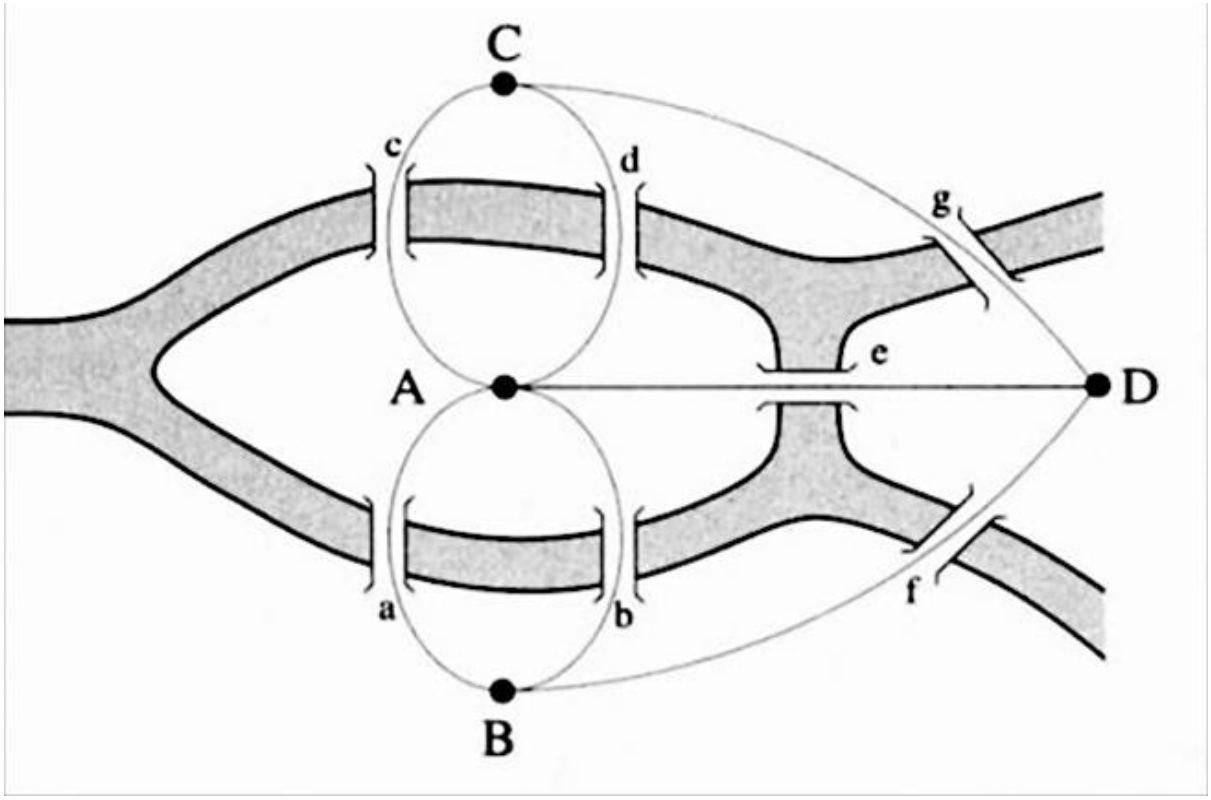


Πηγή: http://social-physics.net/

Η προσέγγιση της αναπαράστασης ενός προβλήματος με γράφημα έγινε για πρώτη φορά τον 18ο αιώνα από τον μαθηματικό Leonard Euler (1707 – 1783) όταν μελέτησε το πρόβλημα των επτά γεφυρών στην πόλη Konigsberg. Το πρόβλημα ήταν εάν κάποιος μπορούσε να περάσει από όλες τις γέφυρες και να γυρίσει στο αρχικό σημείο χωρίς να περάσει από καμία γέφυρα παραπάνω από μια φορά. Πέρασαν 200 περίπου χρόνια, όταν το 1936 εκδόθηκε μια μονογραφία από τον Ούγγρο μαθηματικό Denes Konig. Η θεωρία αναπτύχθηκε σύντομα μετά και τέθηκαν τα θεμέλια της θεωρία των τυχαίων γραφημάτων από τους Ούγγρους μαθηματικούς Paul Erdos (1913 – 1996) και Alfred Renyi (1921 – 1970), στα τέλη της

δεκαετίας του 1950 και θεωρείται η πρώτη αυστηρή και ολοκληρωμένη αντίληψη της μοντέρνας θεωρίας των γραφημάτων.

Εικόνα 19 - Το πρόβλημα των 7 γεφυρών

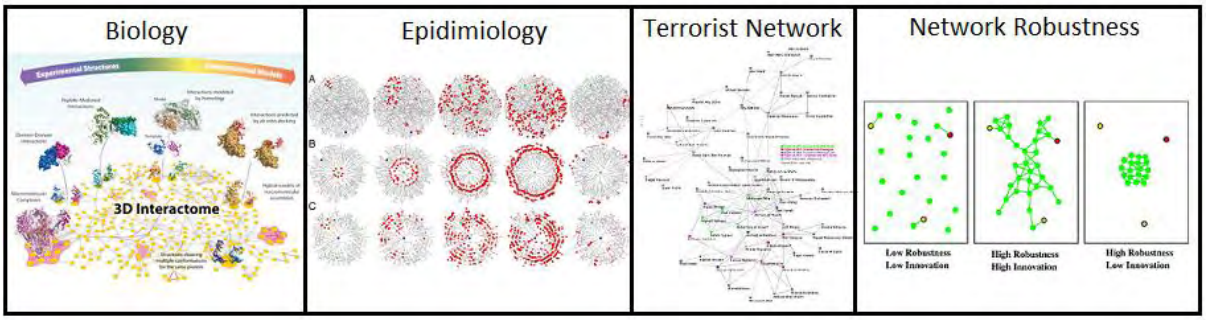


Πηγή: http://physics.weber.edu

Στις μέρες μας, τα σύνθετα δίκτυα είναι μια μίξη πολλών επιστημονικών πεδίων, από την επιστήμη των υπολογιστών και τα εφαρμοσμένα μαθηματικά ως τα οικονομικά, την βιολογία, την κοινωνιολογία, αναλύοντας τα χαρακτηριστικά και τις δομές διαφόρων ομάδων και ειδών προβλημάτων. Εξετάζοντας το συνολικό δίκτυο, τους κόμβους και τους συνδέσμους μεταξύ τους, μπορεί κανείς να αντλήσει σημαντικά συμπεράσματα για την συμπεριφορά του συστήματος. Μερικά από τα αντικείμενα μελέτης των σύνθετων δικτύων είναι οι κεντρικότητες, η εύρεση κοινοτήτων, η εξάπλωση ασθενειών και η ευρωστία των δικτύων. Στην βιολογία μπορεί να χρησιμοποιηθούν τα σύνθετα δίκτυα για την αναπαράσταση πρωτεϊνών και αλληλεπιδράσεων μεταξύ τους, δίκτυα τροφικών αλυσίδων και οικοσυστημάτων, γονιδίων και άλλων. Χρησιμοποιώντας την θεωρία της διάδοσης

ασθενειών υπάρχουν εφαρμογές για ιατρικούς σκοπούς, για την ασφάλεια δικτύων υπολογιστικών συστημάτων και για την διάδοση ιδεών.

Εικόνα 20 - Παραδείγματα χρήσης των σύνθετων δικτύων



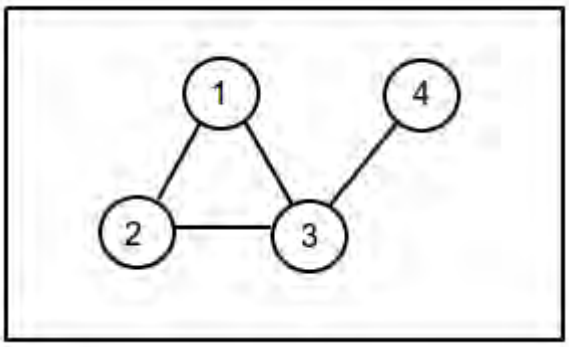
Πηγές: http://sbnb.irbbarcelona.org/, http://currents.plos.org/, http://jtr.st-andrews.ac.uk/, http://www.pnas.org/

Το μεγαλύτερο πείραμα που διεξάχθηκε έγινε στις 21/11/2011 από την Facebook Data Team για το «Small World Project» όπου μελετάται ο μέσος βαθμός διαχωρισμού από τον κάθε ενεργό χρήστη ως προς οποιονδήποτε άλλο. Υπήρχαν 721 εκατομμύρια ενεργοί χρήστες (κορυφές) και 69 δισεκατομμύρια φιλίες μεταξύ τους (ακμές). Τα αποτελέσματα του πειράματος ήταν ότι ο μέσος βαθμός διαχωρισμού ήταν 4,74.

2.2 Γνωστικό υπόβαθρο

Γράφημα είναι η 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. Η παρακάτω εικόνα απεικονίζει ένα απλό, μη κατευθυνόμενο γράφημα.

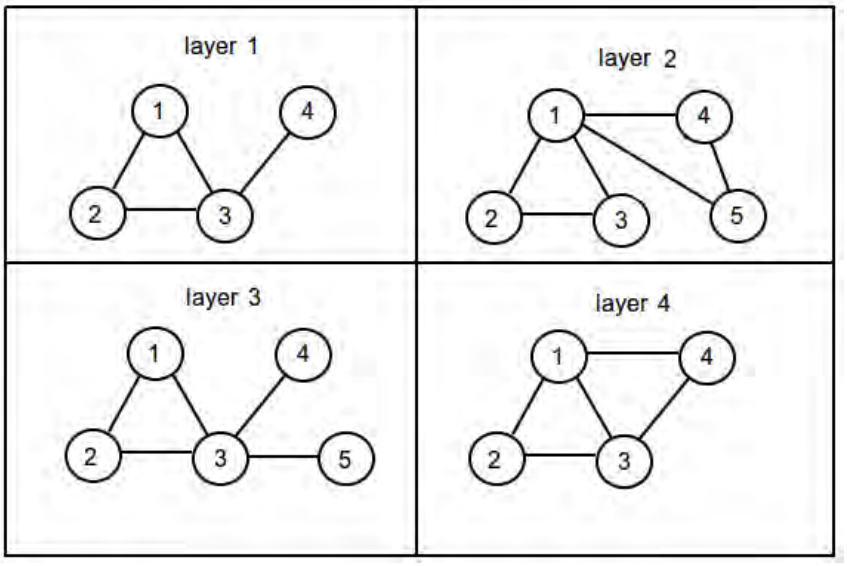
Εικόνα 21 - Απλό, μη κατευθυνόμενο γράφημα



Πηγή: Ιδία επεξεργασία

Πολυδιάστατο γράφημα είναι ένα σύνολο n μονοδιάστατων και ανεξάρτητων γραφημάτων.

Εικόνα 22 – Πολυδιάστατο γράφημα 4 διαστάσεων

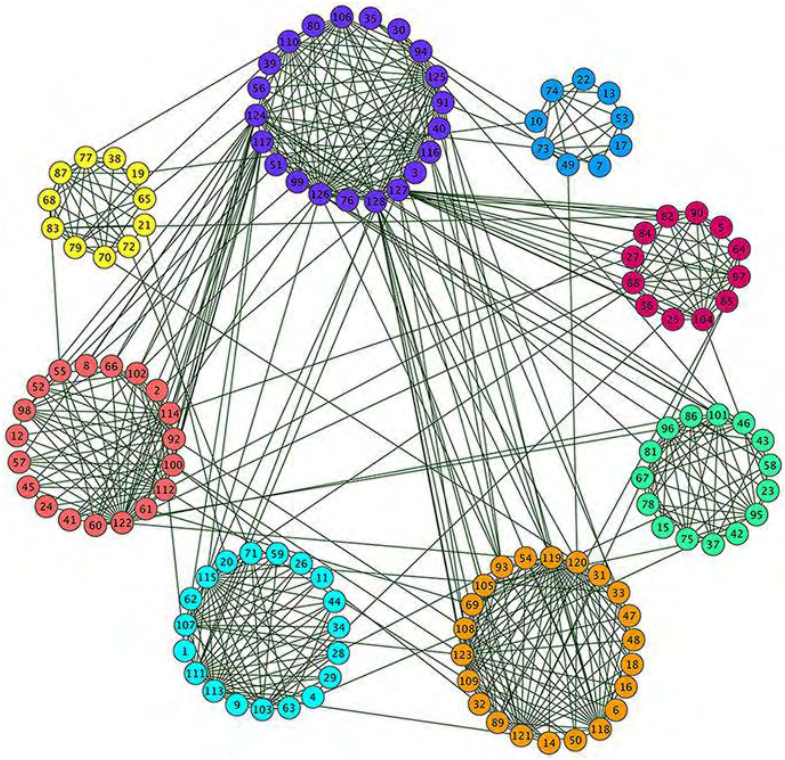


Πηγή: Ιδία επεξεργασία

Ένα παράδειγμα ενός τέτοιου πολυδιάστατου γραφήματος, με 4 διαστάσεις θα μπορούσε να ήταν το εξής: το κάθε layer απεικονίζει ένα κοινωνικό δίκτυο (πχ. Facebook, Tweeter, Instagram, LinkedIn), η κάθε κορυφή (1, 2, 3, 4, 5) απεικονίζει έναν χρήστη και η κάθε ακμή απεικονίζει την φιλία μεταξύ των χρηστών.

Κοινότητα σε ένα γράφημα ορίζεται αφηρημένα ως ένα σύνολο κορυφών που είναι πιο πυκνά συνδεδεμένες μεταξύ τους σε σχέση με τις υπόλοιπες κορυφές του γραφήματος. Οι κοινότητες είναι ομάδες κορυφών που είναι πιθανό να έχουν ομοιότητες ή κοινά χαρακτηριστικά. Η εύρεση κοινοτήτων σαν διαδικασία έχει σκοπό τον διαμοιρασμό του γραφήματος σε τέτοιες ομάδες.

Εικόνα 23 - Γράφημα χωρισμένο σε κοινότητες



Πηγή: http://www.nature.com/

3. Ανασκόπηση

3.1 Είδη Αλγορίθμων

Η εύρεση κοινοτήτων έχει μελετηθεί εκτενώς στη βιβλιογραφία. Έχουν προταθεί πολλές διαφορετικές προσεγγίσεις για την εύρεση κοινοτήτων σε μονοδιάστατα γραφήματα[4].

- Αλγόριθμοι βασισμένοι στο modularity:

  Το modularity ορίζεται ως το κλάσμα των ακμών που υπάρχουν στην κάθε ομάδα αφαιρώντας το κλάσμα των προσδοκώμενων ακμών αν το γράφημα ήταν τυχαίο. Ο στόχος είναι να γίνει ο διαχωρισμός έτσι ώστε το modularity να είναι το μέγιστο.

- Φασματικοί αλγόριθμοι:

  Οι συγκεκριμένοι αλγόριθμοι κάνουν τον διαχωρισμό χρησιμοποιώντας τα ιδιοδιανύσματα των πινάκων του γραφήματος. Ένας Laplacian πίνακας γραφήματος χρησιμοποιείται τυπικά για τον πίνακα του γραφήματος.

- Αλγόριθμοι ορισμού δομής:

  Η ανακάλυψη κοινοτήτων γίνεται έτσι ώστε να ικανοποιείται κάποια πολύ αυστηρή ιδιότητα που αφορά την δομή. Με άλλα λόγια βρίσκουν κοινότητες που ικανοποιούν θεωρητικούς ορισμούς μιας κοινότητας, όπως k-clique, r-quasi-clique, s-plex.

Στις σύγχρονες εφαρμογές, μια οντότητα είναι συσχετισμένη με πολλαπλές πτυχές σχέσεων. Αυτές οι πτυχές σχέσεων μπορούν να μοντελοποιηθούν σαν ένα multi-layer γράφημα, που αποτελείται από πολλαπλά ανεξάρτητα γραφήματα, όπου το κάθε ένα αναπαριστά μία μόνο πτυχή. Προχωρώντας από τα 1-layer γραφήματα στα πολυδιάστατα, προκύπτουν αρκετές προκλήσεις και δυσκολίες. Η κάθε διάσταση περιέχει σημαντικές πληροφορίες από την δικιά της μεριά. Επομένως, είναι απαραίτητο να βρεθεί ο τρόπος ώστε να είναι δυνατή η αξιοποίηση και η αποτελεσματική συγχώνευση των πληροφοριών. Επιπλέον, η επίδοση παίζει έναν πολύ σημαντικό παράγοντα, διότι ο όγκος των δεδομένων αυξάνεται σε μεγάλο βαθμό. Στο 1-layer μελετάμε ένα δίκτυο, ενώ σε multi-layer μελετάμε ένα δίκτυο από δίκτυα.

3.2 Προγενέστεροι Αλγόριθμοι

Στο τμήμα των αλγόριθμων ορισμού δομής έχουν υλοποιηθεί και προταθεί πολλοί αλγόριθμοι για την αναζήτηση και εύρεση στενά συνδεδεμένων υπογραφημάτων. Παρακάτω αναλύονται συνοπτικά μερικοί από τους σημαντικότερους αλγορίθμους, οι οποίοι είναι προγενέστεροι του αλγορίθμου MiMAG.

Crochet (2005):

Ο αλγόριθμος Crochet προτάθηκε από τους Pei et al. [16] και χρησιμοποιεί τον ορισμό των quasi-cliques για να βρει τα πιο στενά υπογραφήματα. Χρησιμοποιεί αναζήτηση depth-first για την εξερεύνηση του χώρου αναζήτησης. Έχουν υλοποιηθεί κάποιες τεχνικές κλαδέματος και ο αλγόριθμος έχει υλοποιηθεί για την εξόρυξη κοινοτήτων σε μονοδιάστατα γραφήματα.

Cocain (2006):

Ο αλγόριθμος Cocain από τους Zeng et al. [17] χρησιμοποιεί τον ορισμό των quasi-cliques για την ανακάλυψη κοινοτήτων στο δίκτυο που εξετάζει. Χρησιμοποιώντας κάποιες από τις ιδιότητες των quasi-cliques κλαδεύει κορυφές και ακμές για να μικρύνει ο χώρος αναζήτησης και να επιταχυνθεί η διαδικασία αναζήτησης. Έχει υλοποιηθεί για γραφήματα με ένα layer.

Quick (2008):

Οι Liu et al. [11] στον αλγόριθμο Quick χρησιμοποιούν τις quasi-cliques και την αναπαράσταση με τη μορφή δέντρου. Κάνει χρήση του depth-first για την εξερεύνηση στον χώρο αναζήτησης. Έχουν εφαρμοστεί πολλές τεχνικές κλαδέματος σε σχέση με προγενέστερους του αλγορίθμους για την μείωση του χώρου αναζήτησης, επιδιώκοντας καλύτερους χρόνους επίδοσης. Έχει υλοποιηθεί για την αναζήτηση κοινοτήτων σε ένα layer.

CoPaM (2009):

Ο αλγόριθμος CoPaM από τους Moser et al. [18] χρησιμοποιεί και αυτός το μοντέλο των quasi-cliques. Ξεκινάει με μια προ επεξεργασία των δεδομένων όπου αφαιρεί τις ακμές και τις κορυφές που δεν θα μπορέσουν ποτέ να συμπεριληφθούν στο αποτέλεσμα. Χρησιμοποιεί ένα μοντέλο πλεονάζουσας σχέσης για να αποφεύγει να δημιουργεί όμοια υπογραφήματα κατά τη διάρκεια της εκτέλεσης του. Ο αλγόριθμος CoPaM έχει υλοποιηθεί για πολυεπίπεδα γραφήματα.

GAMer (2010):

Ο αλγόριθμος GAMer από τους S. Gün*n*emann et al. [18] χρησιμοποιεί και αυτός τον ορισμό των quasi-cliques για να βρει ομοιογενείς ομάδες κορυφών σε γραφήματα δικτύων. Αξιοποιεί και δίνει έμφαση στην πυκνότητα, το μέγεθος και τον αριθμό των ενεργών layer του κάθε cluster για να συλλέξει τα πιο ενδιαφέροντα. Επίσης, χρησιμοποιεί ένα μοντέλο πλεονασμού για να απορρίπτει τα πλεονάζοντα ή διπλότυπα cluster. Έτσι, το αποτέλεσμα περιέχει τα πιο ενδιαφέροντα και μη πλεονάζοντα cluster. Χρησιμοποιεί τεχνικές κλαδέματος για την βελτίωση της απόδοσης και είναι υλοποιημένος για πολύ-επίπεδα γραφήματα.

4. Μοντελοποίηση

4.1 Σύμβολα που θα χρησιμοποιηθούν

G :	Γράφημα
V :	Σύνολο κορυφών του γραφήματος
E :	Σύνολο ακμών του γραφήματος
O :	Σύνολο κορυφών σε ένα cluster
$\gamma G_i(O)$:	Πυκνότητα του O στο layer i
S :	Το σύνολο των layer όπου το O είναι γ-quasi-clique
$\gamma_S(O)$:	Ο μέσος όρος της πυκνότητας του O
$cand_O$:	Το σύνολο των υποψήφιων κορυφών προς επέκταση του συνόλου κορυφών O
So :	Τα ενεργά επίπεδα ενός κόμβου O

4.2 Μοντέλο MLCS

Το μοντέλο MLCS[5] (Multi-Layer Coherent Subgraph) ανακαλύπτει ομάδες κόμβων που είναι πυκνά συνδεδεμένες με ακμές, σε ένα υποσύνολο των layers. Αυτές οι ομάδες κόμβων ονομάζονται (multi-layer) συνδεδεμένα υπογραφήματα. Πρέπει να επισημανθεί ότι τα υπογραφήματα δεν είναι απαραίτητο να εμφανίζονται σε όλα τα layer, αλλά σε ένα υποσύνολο των layer. Επομένως, για κάθε συνδεδεμένο υπογράφημα βρίσκουμε και ένα ανεξάρτητο σύνολο από σχετικά layer.

Επιπλέον, οι κόμβοι μπορούν να ανήκουν σε περισσότερα από ένα συνδεδεμένα υπογραφήματα. Όμως, επιτρέποντας την επικάλυψη, μπορεί να εμφανιστεί ένας πολύ μεγάλος αριθμός από έγκυρα υπογραφήματα τα οποία αναπαριστούν πλεονάζουσες πληροφορίες. Για να λυθεί αυτό το πρόβλημα, επιτρέπεται η επικάλυψη ως ένα σημείο. Το τελικό αποτέλεσμα περιέχει τα πιο ενδιαφέροντα υπογραφήματα, σε σχέση με μια συνάρτηση ποιότητας.

Χρησιμοποιείται best-first αναζήτηση για να βρεθεί μια προσεγγιστική λύση. Ο αλγόριθμος αναζήτησης best-first είναι μια εδραιωμένη αρχή αναζήτησης για την εξερεύνηση γραφημάτων. Εδώ, η αναζήτηση ξεκινά από έναν αρχικό κόμβο (root node) και επεκτείνεται επαναληπτικά ως προς τον πιο “υποσχόμενο” κόμβο. Τα πιο υποσχόμενα υπογραφήματα επεκτείνονται ώστε να βρεθούν πρώτα τα πιο ενδιαφέροντα cluster.

Ορισμός 1 (Multi-layer γράφημα):

Ένα πολυεπίπεδο γράφημα G για ένα σύνολο επιπέδων Dim = {1, ..., d} είναι ένα σύνολο $G = \{G_i \mid i \in Dim\}$ *από γραφήματα.*

$$G_i = (V, E_i), E_i \subseteq V \times V : E_i \rightarrow R$$

όπου κάθε επίπεδο του γραφήματος G_i , $i \in Dim$ *είναι ένα μη κατευθυνόμενο γράφημα χωρίς self-loops .*

Στην περίπτωση που έχουμε διαφορετικά σύνολα κορυφών ανάμεσα σε διαφορετικά επίπεδα V_i μπορούμε να θεωρήσουμε ως σύνολο κορυφών την ένωση $V = U V_i$.

4.3 Μοντέλο των cluster

Αρχικά, θεωρούμε πως έχουμε ένα μοναδικό επίπεδο G_i . Για την πυκνότητα της συστάδας χρησιμοποιούμε το μοντέλο quasi-clique. Το quasi-clique μοντέλο ορίζει πυκνά διασυνδεδεμένα υπογραφήματα βασισμένα στις διασυνδέσεις εντός της συστάδας.

Ορισμός 2 (γ – QUASI – CLIQUE):

Ένα σύνολο κορυφών $O \subseteq V$ σε ένα γράφημα $G = (V, E)$ αποτελεί μια γ-quasi-clique για ένα $\gamma \in [0, 1]$ αν :

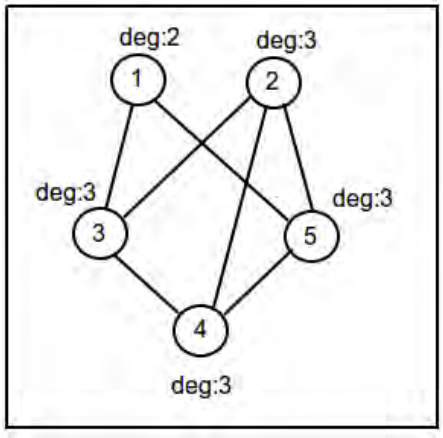
$$\forall\, \upsilon \in O : deg_G^O(\upsilon) \geq \lceil \gamma \cdot (|O| - 1) \rceil$$

όπου $deg_G{}^O(v) = |\{u \in O \mid (u, v) \in E\}|$. Η πυκνότητα μιας γ-quasi-clique O σε ένα επίπεδο γραφήματος G_i ορίζεται ως:

$$\gamma_{Gi}(O) = \frac{min_{\upsilon \in O}\{deg_{Gi}^O(\upsilon)\}}{|O|-1}$$

Μια quasi-clique είναι ένα υπογράφημα το οποίο ικανοποιεί ένα ορισμένο από τον χρήστη κάτω όριο βάρους κορυφής. Για το μοντέλο των cluster που θα χρησιμοποιήσουμε, θεωρούμε ένα σύνολο κορυφών ως πυκνό αν είναι 0,5-quasi-clique, δηλαδή αν η πυκνότητα της quasi-clique είναι τουλάχιστον 0,5. Αυτό εξασφαλίζει ότι το υπογράφημα θα είναι συνδεδεμένο στο γράφημα.

Εικόνα 24 - Παράδειγμα υπογραφήματος

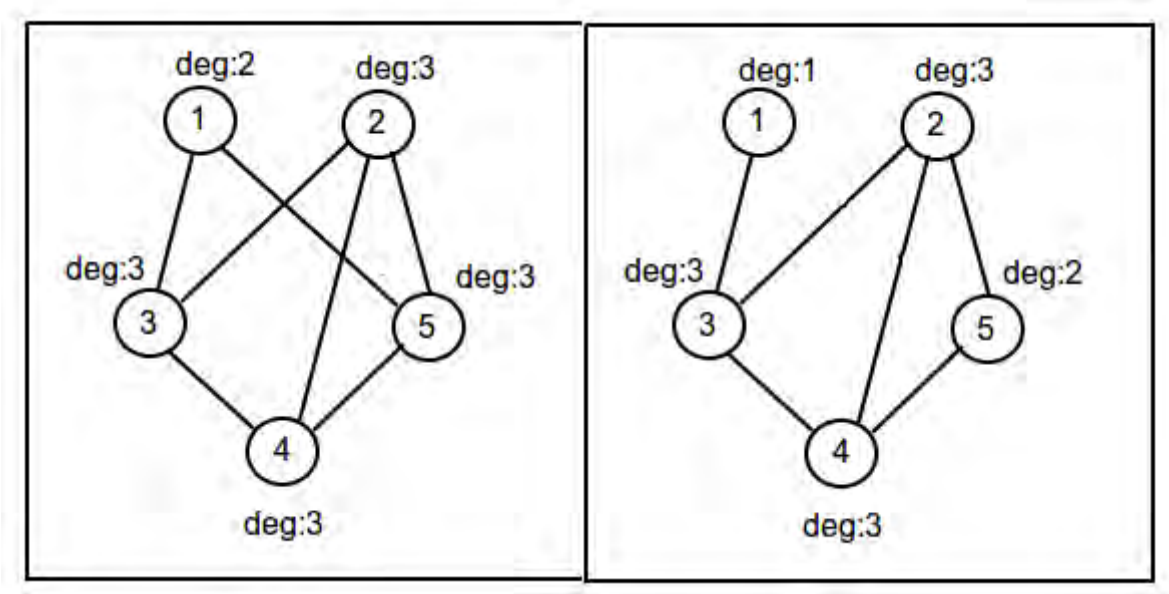


Πηγή: Ιδία επεξεργασία

για γ = 0,5 και για 5 κορυφές ($|O| = 5$):
deg(v) ≥ ceil(γ · ($|O|$ - 1)) ⇒ deg(v) ≥ ceil(0,5 · (5 - 1)) ⇒ deg(v) ≥ 2
που σημαίνει πως η κάθε κορυφή πρέπει να συνδέεται με τουλάχιστον δύο κορυφές.
Η αλλιώς, πρέπει $\gamma_{Gi}(O) \geq 0,5$

Παράδειγμα:

Εικόνα 25 - Υπογραφήματα παραδείγματος



Πηγή: Ιδία επεξεργασία

Για το αριστερό υπογράφημα:

$$\gamma_{Gi}(O) = \frac{min_{v \in O}\{deg^{O}_{Gi}(v)\}}{|O|-1} => \frac{2}{4} = 0,5$$

Είναι γ-quasi-clique γιατί deg(v) ≥ 2, για κάθε $v \in V$

Για το δεξί υπογράφημα:

$$\gamma_{Gi}(O) = \frac{min_{v \in O}\{deg^{O}_{Gi}(v)\}}{|O|-1} => \frac{1}{4} = 0,25$$

Δεν είναι γ-quasi-clique γιατί deg(1) ⩽ 2

Ορισμός 3 (Μέγιστη γ-quasi-clique):

Δεδομένου ενός γραφήματος G = (V, E) και ένα σύνολο κορυφών X, όπου X ⊆ Vτο G(X) είναι μια μέγιστη γ-quasi-clique αν δεν υπάρχει άλλο σύνολο κορυφών Y όπου Y ⊃ X και G(Y) είναι quasi-clique.

Συνεχίζοντας, αναζητούμε cluster στα διάφορα layer ενός multi-layer γραφήματος. Επομένως, η συστάδα πρέπει να ικανοποιεί την μονοδιάστατη ιδιότητα για γ-quasi-clique σε κάθε ένα από το υποσύνολο των layer του (συνολικού) multi-layer γραφήματος. Αν μια ακμή (u, υ) υπάρχει σε ένα επίπεδο, δεν σημαίνει ότι θα υπάρχει και σε κάποιο άλλο. Εξετάζουμε, δηλαδή, το ίδιο σύνολο κορυφών σε διαφορετικά layer όπου οι ακμές και η μορφολογία μπορεί να μην είναι η ίδια. Συνεπώς, στο μοντέλο μας πρέπει να εξετάσουμε την πυκνότητα του συνδεδεμένου υπογραφήματος που εξετάζουμε ξεχωριστά σε κάθε layer.

Ορισμός 4 (MLCS cluster):

Ένα MLCS cluster C = (O, S) σε ένα multi-layer γράφημα G = {G_i | i ∈ Dim} αποτελείται από ένα σύνολο κορυφών O ⊆ Vκαι ένα όχι κενό σύνολο από σχετικά layer S ⊆ Dim έτσι ώστε $\forall i \in Dim$*: i ∈ S ⇔ το O είναι ένα MLCS cluster στο γράφημα του layer* G_i .

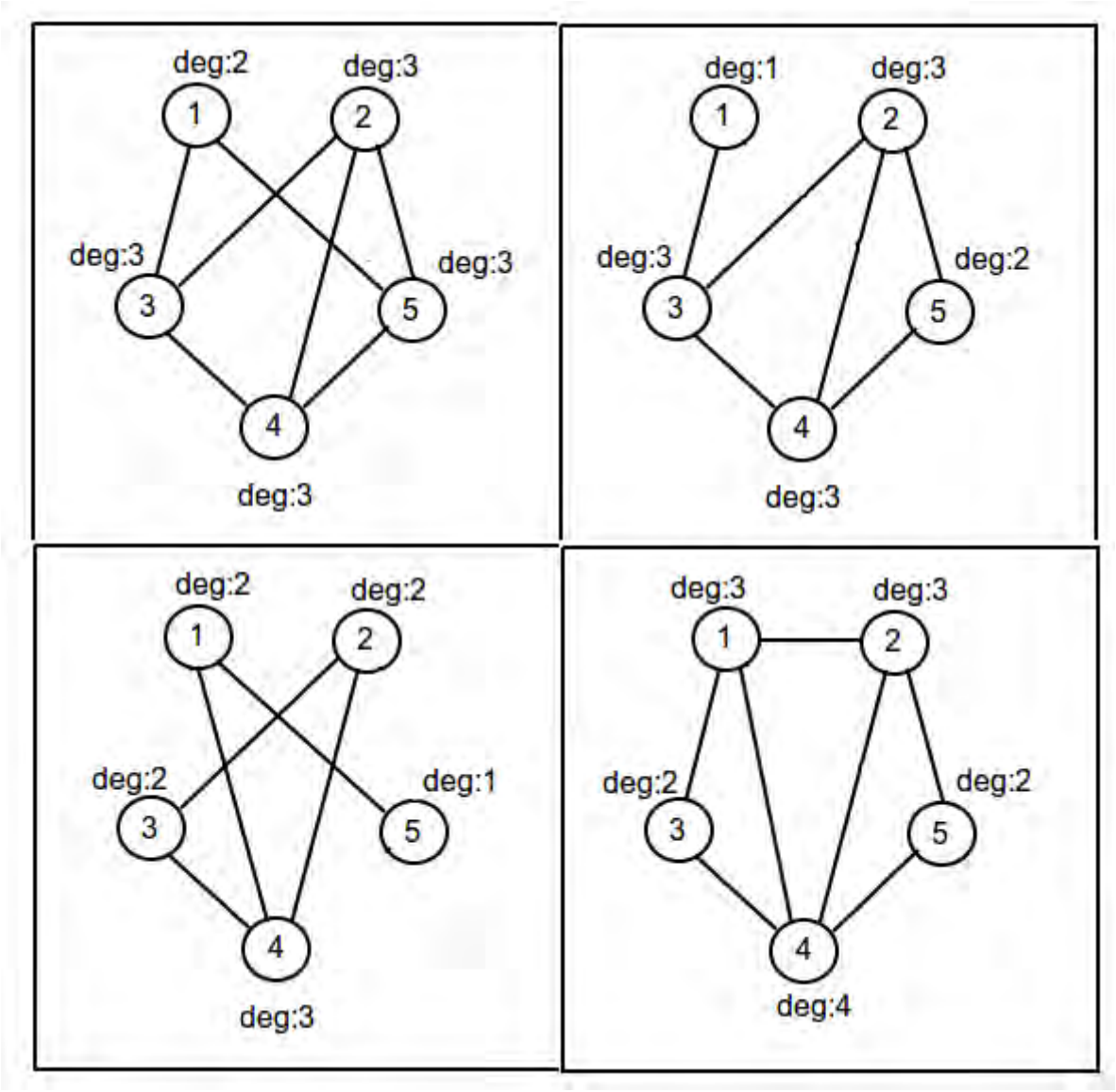
Η πυκνότητα ενός cluster C = (O, S) ορίζεται ως

$$\gamma_S(O) = \frac{1}{|S|} \sum_{i \in S} \gamma G_i(O)$$

Αφού οι ακμές διαφέρουν από το ένα layer στο άλλο. διαφέρει και η πυκνότητα ενός cluster ανάμεσα στα layer. Έτσι, ορίζουμε την πυκνότητα ενός cluster ως το μέσο όρο της πυκνότητας του υποσυνόλου των layer στα οποία δημιουργείται γ-quasi-clique, δηλαδή $\gamma_{Gi} \geq 0{,}5$.

Παράδειγμα:

Εικόνα 26 - Υπογράφηματα παραδείγματος



Πηγή: Ιδία επεξεργασία

O = {1, 2, 3, 4, 5}

$\gamma_{G1} = 0,5$, $\gamma_{G2} = 0,25$, $\gamma_{G3} = 0,25$, $\gamma_{G4} = 0,5$

Σύμφωνα με τα παραπάνω, το σύνολο κορυφών O είναι γ-quasi-clique στα layers: {1, 4} και άρα το cluster έχει S = {1, 4} και $\gamma_S(O) = ½$.

4.4 Ποιότητα ενός cluster

Το πόσο ενδιαφέρον είναι κάποιο cluster προσδιορίζεται από μια συνάρτηση ποιότητας που μπορεί να οριστεί από τον χρήστη. Συνήθως, ένα cluster όσο περισσότερους κόμβους περιέχει τόσο πιο ενδιαφέρον είναι. Για αυτό τον λόγο, οι περισσότερες προσεγγίσεις στην εξόρυξη των quasi-cliques από ένα γράφημα, στοχεύουν στο να βρουν τις μέγιστες quasi-cliques όσον αφορά τις κορυφές που περιέχουν. Ωστόσο, αν απλά μεγιστοποιήσουμε τον αριθμό των κορυφών που περιέχει ένα cluster, μπορεί να οδηγηθούμε στην ανακάλυψη τέτοιων cluster που να ανήκουν σε λίγα layer ή που να μην έχουν την αποδεκτή πυκνότητα. Έτσι λοιπόν, η συνάρτηση ποιότητας καθίσταται αναγκαία και είναι ένα trade-off ανάμεσα στο μέγεθος ενός cluster, στον αριθμό των layer στα οποία ανήκει και στην πυκνότητα του.

Δεν ενδιαφερόμαστε για cluster τα οποία είναι πολύ μικρά (που περιέχουν λιγότερες από 8 κορυφές) ή για αυτά που ανήκουν σε λιγότερα από δύο layer.

Ορίζουμε την ποιότητα ενός cluster C = (O, S) ως:

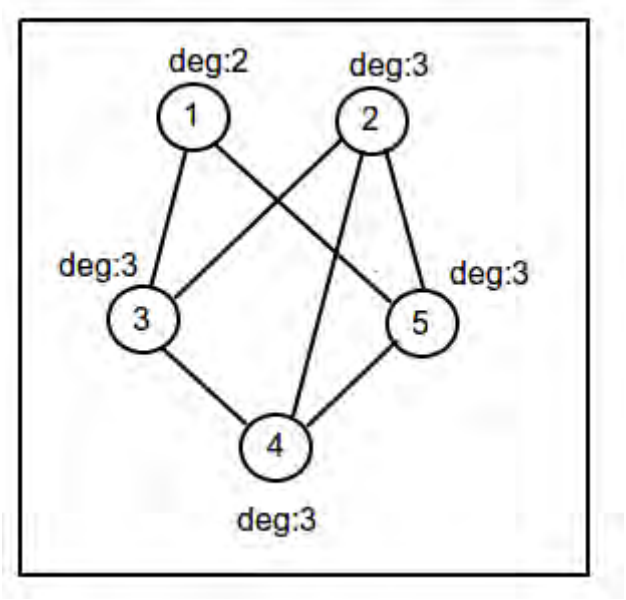
$$Q(C) = \begin{cases} |O| \cdot |S| \cdot \gamma_S(O) & |O| \geq 8 \wedge |S| \geq 2 \\ -1 & \text{else} \end{cases}$$

Στα cluster τα οποία δεν θεωρούνται ενδιαφέροντα αναθέτετε η τιμή -1 και δεν θα ανήκουν ποτέ σε ένα MLCS cluster αφού θα μειώνουν την συνολική ποιότητα του cluster. Μολονότι στο πειραματικό κομμάτι χρησιμοποιήθηκε η παραπάνω συνάρτηση ποιότητας, σε άλλες εφαρμογές θα μπορούσε να αλλαχτεί, ώστε να αναζητηθούν και να ανακαλυφθούν διαφορετικού είδους cluster με διαφορετικό μέγεθος, πυκνότητας ή layer στα οποία ανήκει.

4.5 Μοντέλο ομαδοποίησης κορυφών

Στα προηγούμενα κεφάλαια, εξετάστηκαν τα χαρακτηριστικά που πρέπει να πληροί ένα cluster. Τα cluster είναι δυνατό να επικαλύπτονται σε κάποια σημεία αναμεταξύ τους. Με το να συλλέγουμε, όμως, όλα τα έγκυρα cluster που ανακαλύπτονται οδηγούμαστε σε έναν μεγάλο αριθμό αποτελεσμάτων τα οποία είναι σε σημαντικό βαθμό όμοια και συνεπώς αναπαριστούν πλεονάζουσα πληροφορία. Το τελικό αποτέλεσμα πρέπει να περιέχει cluster τα οποία είναι τα πιο "ενδιαφέροντα" και να μην είναι πλεονάζοντα.

Εικόνα 27 - Υπογράφημα παραδείγματος



Πηγή: Ιδία επεξεργασία

Ένα παράδειγμα από cluster που επικαλύπτονται είναι:

C1 = (O = {1, 2, 3, 4, 5}, S = {1}) με Q = 0,66

C2 = (O = {2, 3, 4, 5}, S = {1}) με Q = 0,5

Τα cluster C1 και C2 επικαλύπτονται. Με την προϋπόθεση ότι αναζητούμε τα cluster με την καλύτερη ποιότητα, χωρίς να μας ενδιαφέρει ο αριθμός των κορυφών μπορούμε να συμπεράνουμε πως το cluster C2 είναι πλεονάζον διότι περιέχει πληροφορία που βρίσκεται στο C1 και έχει μικρότερη ποιότητα από το cluster C1.

4.6 Πλεονάζουσα σχέση ανάμεσα σε cluster

Για να αποφύγουμε τα πλεονάζοντα cluster είναι επιτακτικό να υπάρχει μια σχέση πλεονασμού. Ορίζουμε ένα cluster C ότι είναι πλεονάζον σχετικά με ένα άλλο cluster C' όταν ένα σημαντικό μέρος από τις ακμές του C ανήκει και στο C' (οπότε και αναπαριστούν μερικώς ίδια πληροφορία) και η ποιότητα του C' είναι μεγαλύτερη από του C.

Ορισμός 5 (Πλεονάζουσα σχέση):

Ένα cluster C = (O, S) είναι πλεονάζον σε σχέση με ένα άλλο cluster C' = (O', S') (σε συντομογραφία: $C \preccurlyeq_{red} C'$) εάν:

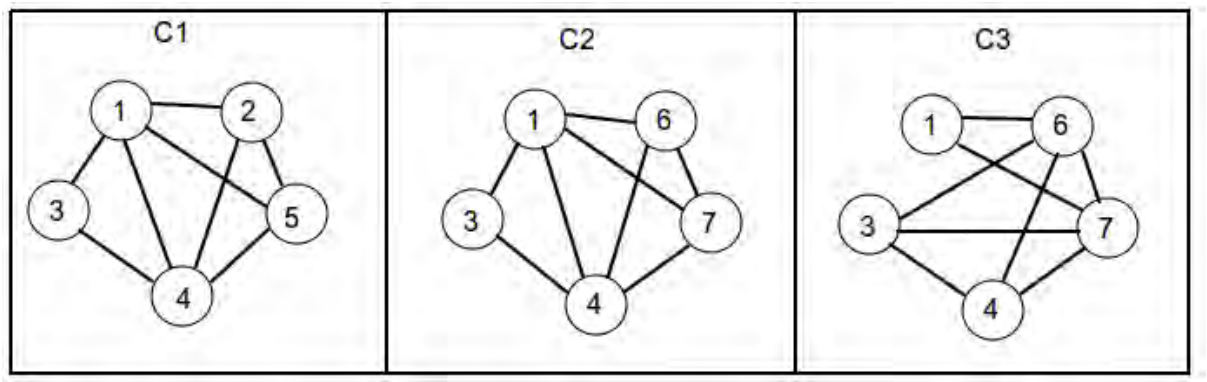
$$C \neq C' \wedge Q(C) \leq Q(C') \wedge \frac{1}{|S|} \sum_{i \in S \cap S'} \frac{|E_i(O) \cap E_i(O')|}{|E_i(O)|} \geq r$$

για παράμετρο πλεονασμού $r \in (0, 1]$.

Η προκαθορισμένη (default) τιμή της παραμέτρου πλεονάζουσας πληροφορίας είναι r = 0.25 .

Παράδειγμα:

Εικόνα 28 - Cluster παραδείγματος



Πηγή: Ιδία επεξεργασία

Στην παραπάνω εικόνα έχουμε 3 cluster (C1, C2, C3) με ίδιες ποιότητες Q1 = Q2 = Q3 = 1/2.

Χρησιμοποιώντας τον τύπο

$$\frac{1}{|S|}\sum_{i \in S \cap S'} \frac{|E_i(O) \cap E_i(O')|}{|E_i(O)|}$$

για να βρούμε τα cluster C2 και C3 είναι πλεονάζον σε σχέση με το C1.
Έχουμε για το C2: 3/8 = 0,375 > r = 0.25 ⇒ το C2 είναι πλεονάζον ως προς το C1
Έχουμε για το C3: 2/9 = 0,22 < r = 0.25 ⇒ το C3 δεν είναι πλεονάζον ως προς το C1

Βρήκαμε δηλαδή, πως το cluster C2 περιέχει παρόμοια πληροφορία με το cluster C1 σε τέτοιο βαθμό ώστε να το θεωρήσουμε πλεονάζον και να το απορρίψουμε. Αντίθετα, το cluster C3 που είναι σε ένα κομμάτι του αλληλεπικαλυπτόμενο με το cluster C1, προκύπτει πως δεν επικαλύπτεται σε μεγάλο βαθμό και μπορεί να γίνει αποδεκτό.

Με την παραπάνω σχέση πλεονασμού μπορούμε να επιλέξουμε μόνο τα cluster που έχουν χρήσιμη πληροφορία και ταυτόχρονα αυτά που έχουν την μέγιστη ποιότητα.

Ορισμός (MLCS clustering):

Δοσμένου ενός multi-layer γραφήματος G και του συνόλου A όλων των έγκυρων MLCS cluster, η μέγιστης ποιότητας ομαδοποίηση Result ⊂ A ικανοποιεί:

- *Μη πλεονασμό:* $\neg \exists C, C \in Result : C \prec_{red} C$
- *Μέγιστο άθροισμα ποιότητας:* $\neg \exists Result \subseteq A$*: Result' δεν έχει πλεονάζοντα στοιχεία και* $\Sigma_{C \in Result'} Q(C) > \Sigma_{C \in Result} Q(C)$.

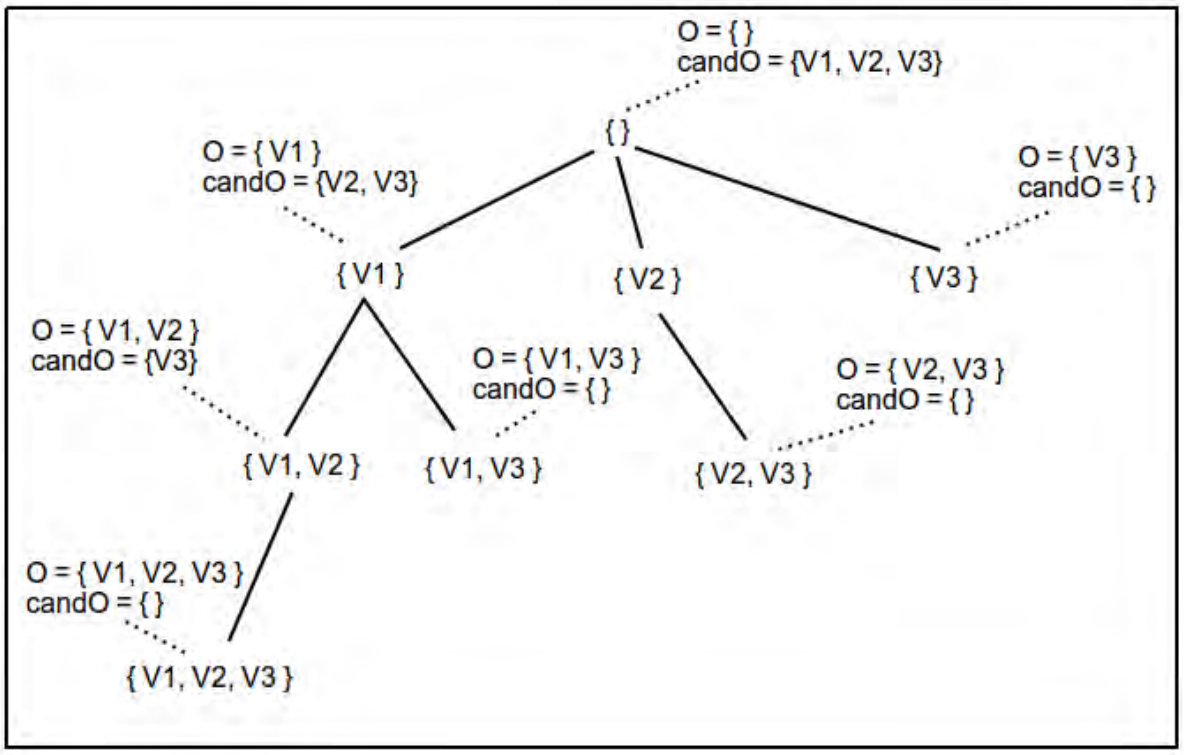
5. MiMAG

5.1 Ο αλγόριθμος MiMAG

Ο αλγόριθμος MiMAG[5] (Mining Multi-layered Attributed Graphs) προσπαθεί να βρει μια προσεγγιστική λύση. Σκοπός είναι, φτάνοντας στο τέλος να έχουμε ομαδοποιήσει τα cluster με τέτοιο τρόπο, ώστε να είναι αδύνατο να προστεθεί κάποιο cluster στο τελικό αποτέλεσμα με Q > 0 χωρίς να παραβιάζει τους κανονισμούς περί πλεοναζόντων cluster.

Ο MiMAG είναι μερικώς βασισμένος στον Quick Algorithm[11] για την εύρεση των quasi-cliques. Στον Quick Algorithm τα σύνολα των κόμβων $O \subseteq V$ απαριθμούνται από μια αναζήτηση κατά βάθος, στο δέντρο απαρίθμησης του συνόλου των κόμβων (set enumeration tree) για κάθε layer i.

Εικόνα 29 - Set enumeration tree



Πηγή: Ιδία επεξεργασία

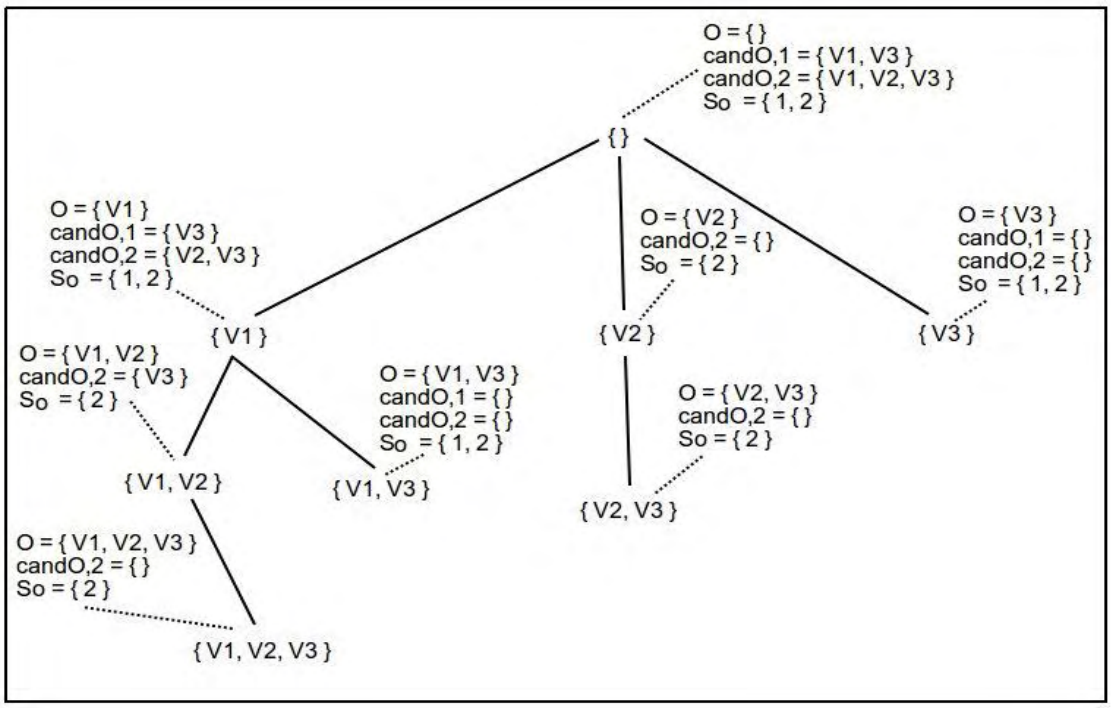
Να γίνει η διευκρίνηση πως όταν λέμε κόμβος, αναφερόμαστε στους κόμβους του set enumeration tree ως σύνολο κορυφών. Κάθε κόμβος που ανακαλύπτεται εξετάζεται εάν είναι quasi-clique. Επίσης, κάθε κόμβος είναι συσχετισμένος με ένα σύνολο υποψηφίων $cand_O$ το οποίο περιέχει όλες τις κορυφές που είναι ταξινομημένες πίσω από τις κορυφές του O σε μια συγκεκριμένη σειρά < . Ένα παιδί κόμβος O' επεκτείνει τον γονιό κόμβο O προσθέτοντας μια κορυφή από το σύνολο $cand_O$. Το set enumeration tree περιέχει όλα τα πιθανά σύνολα κορυφών $O \subseteq V$.

Μια απλοϊκή προσέγγιση για να βρούμε τα cluster που μας ενδιαφέρουν θα ήταν με μια συγχρονισμένη διάσχιση των δέντρων. Θα πρέπει να εφαρμοστεί ο αλγόριθμος Quick σε κάθε ένα από τα επίπεδα του γραφήματος ξεχωριστά ώστε να βρεθούν όλα τα μονοδιάστατα cluster. Έπειτα, πρέπει να ενωθούν τα επιμέρους αποτελέσματα για να ανακαλυφθούν τα cluster που ανήκουν σε παραπάνω από ένα layer. Τέλος, αφαιρούνται τα πλεονάζοντα cluster και έχουμε σαν αποτέλεσμα τα έγκυρα, multi-layer, μη πλεονάζοντα cluster.

Η παραπάνω προσέγγιση μπορεί να βελτιωθεί με μια συγχρονισμένη διάσχιση όλων των set enumeration trees ταυτόχρονα, δηλαδή όλα τα στιγμιότυπα των δέντρων (όλων των layer) να διασχίζονται την ίδια στιγμή. Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας ένα extended set enumeration tree. Κάθε κόμβος O έχει ένα σύνολο ενεργών layer S_o (τα ενεργά layer αναπαριστούν το σύνολο των layer στα οποία ένας κόμβος O δεν έχει κλαδευτεί από το set enumeration tree) και ένα σύνολο από υποψήφιες κορυφές ως προς επέκταση $cand_{O,i}$, για $i \in S_o$. Κατά τη διάσχιση, για κάθε κόμβο O που επισκεπτόμαστε, ελέγχουμε αν σχηματίζει ένα MLCS cluster σε ένα υποσύνολο των ενεργών layer.

Πρέπει να επισημανθεί ότι το σύνολο των ενεργών επιπέδων S_O ενός κόμβου O και το υποσύνολο S ενός πιθανού cluster C = (O, S) είναι διαφορετικά σύνολα. Ισχύει ότι $S \subseteq S_O$ αλλά αυτά τα δύο σύνολα δεν είναι απαραίτητα ίσα. Ακόμα και αν είναι ενεργό κάποιο layer, δεν σημαίνει απαραίτητα πως σχηματίζεται cluster στο συγκεκριμένο layer (μπορεί πχ. να μην εκπληρώνει την απαραίτητη πυκνότητα σε αυτό το layer).

Εικόνα 30 – Extended set enumeration tree με 3 κορυφές και 2 layer



Πηγή: Ιδία επεξεργασία

Για την ενημερωμένη (Informed) best-first διάσχιση πρέπει αντί να δημιουργήσουμε πρώτα όλα τα cluster, να αφήσουμε τα τελικά και όχι πλεονάζοντα cluster να προστίθενται στο αποτέλεσμα διαδοχικά. Εφόσον θέλουμε να μεγιστοποιήσουμε την ποιότητα των cluster του αποτελέσματος, στοχεύουμε στην δημιουργία των cluster σε φθίνουσα σειρά με βάση την ποιότητα τους και προσθέτουμε στο σύνολο του αποτελέσματος τα μη πλεονάζοντα cluster με την μεγαλύτερη ποιότητα. Έτσι, με informed best-first διάσχιση υπολογίζουμε μια εκτίμηση της ποιότητας του κάθε κόμβου με σύνολο κορυφών O, η οποία μας προσφέρει ένα άνω όριο για την ποιότητα κάθε cluster που μπορεί να βρεθεί σε ένα υποδέντρο του κόμβου με σύνολο κορυφών O. Ξεκινώντας τη διάσχιση από τον κόμβο της ρίζας (root), σε κάθε βήμα επεκτείνουμε τον κόμβο με σύνολο κορυφών O με τη μεγαλύτερης ποιότητας κορυφή που έχει εκτιμηθεί (δηλαδή κατεβαίνει κατά ένα βήμα στο υποδέντρο που προέρχεται από το O με την μεγαλύτερη ποιότητα).

Ένα σημαντικό πράγμα που πρέπει να ληφθεί υπόψη είναι ότι ακόμα και αν ένα cluster C βρεθεί σε έναν κόμβο που μόλις έχει επεκταθεί δεν μπορεί να μπει στο αποτέλεσμα απευθείας. Επειδή η εκτίμηση της ποιότητας θέτει ένα άνω όριο της ποιότητας του υποδέντρου, το C μπορεί να έχει μικρότερη ποιότητα. Μπορεί, δηλαδή, να υπάρχουν άλλα υποδέντρα (και πιθανά cluster) με μεγαλύτερες (εκτιμώμενες) ποιότητες. Έτσι λοιπόν, ο

MiMAG διατηρεί μια ουρά προτεραιότητας η οποία περιέχει όλα τα υποδέντρα που περιμένουν να επεξεργαστούν καθώς και όλα τα cluster που έχουν ανακαλυφθεί, τα οποία δεν μπορούσαν να προστεθούν στο σύνολο του αποτελέσματος. Η ουρά αυτή ταξινομείται ως προς την (εκτιμώμενη) ποιότητα των υποδέντρων και των cluster. Αν το πρώτο στοιχείο της ουράς είναι ένα cluster, τότε μπορούμε με βεβαιότητα να συμπεράνουμε πως δεν υπάρχουν καλύτερα cluster. Σε αυτήν την περίπτωση, ελέγχουμε εάν το cluster είναι πλεονάζον και αν δεν είναι, μπορούμε να το προσθέσουμε στο αποτέλεσμα.

Σε μια ουρά, ένα υποδέντρο (ST) αναπαρίσταται από μια πλειάδα $ST = (O, S_O, \{cand_{O,i} \mid i \in S_O\})$ όπου το O είναι το σύνολο των κορυφών στον αρχικό κόμβο του υποδέντρου, το S_O είναι το σύνολο των ενεργών layer για το O και το $cand_{O,i}$ είναι το σύνολο των υποψηφίων ως προς επέκταση. Η ποιότητα ενός υποδέντρου Qest(ST) δηλώνει ένα άνω όριο για την ποιότητα των cluster σε αυτό το υποδέντρο.

Εικόνα 31 - Αλγόριθμος MiMAG

Algorithm 1 MiMAG: Best-first search for MLCS clusters

Require: ML-Graph $\mathcal{G} = \{G_i \mid i \in Dim\}$ with $G_i = (V, E_i, l_i)$
Ensure: Redundancy-free, maximal clustering *Result*

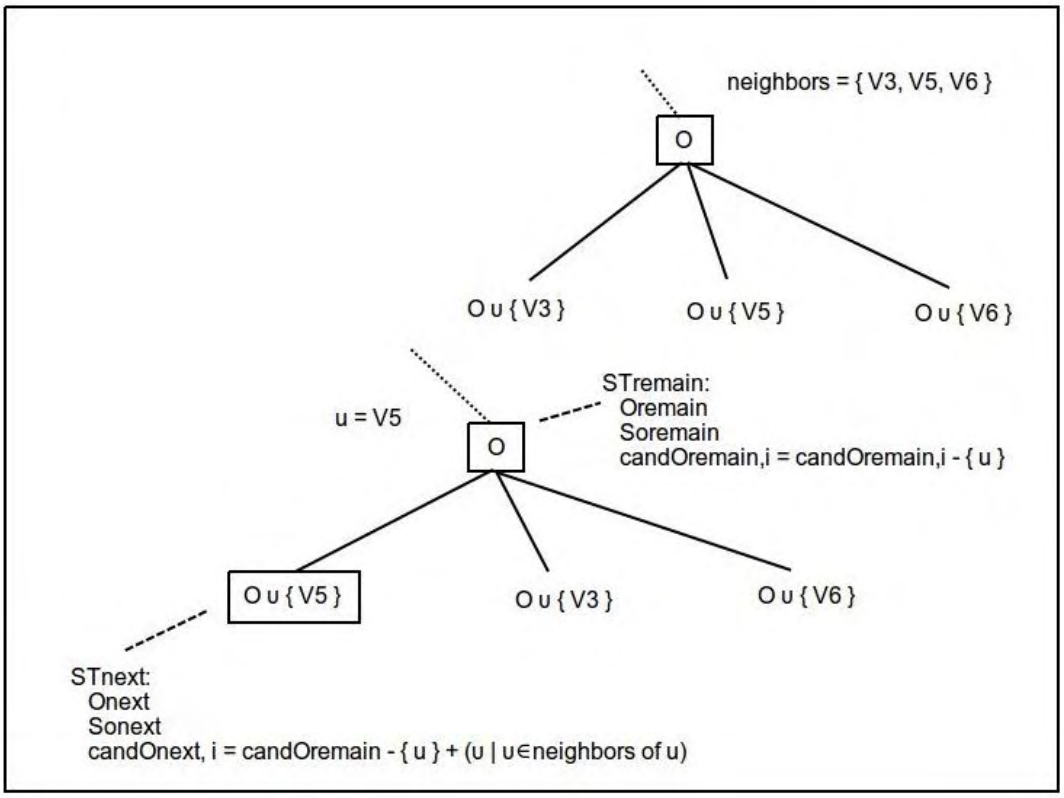
1: $Result := \emptyset$
2: $queue := \{(\emptyset, Dim, \{cand_{\emptyset,i} = V \mid i \in Dim\})\}$
3: **while** $queue \neq \emptyset$ **do**
4: $Obj := queue.pop()$
5: **if** Obj is cluster $C = (O, S)$ **then**
6: **if** $\neg\exists C' \in Result : C \prec_{red} C'$ **then** $Result.add(C)$
7: **else** ▷ Obj is $ST = (O, S_O, \{cand_{O,i} \mid i \in S_O\})$
8: $neighbors := \bigcup_{i \in S_O} \{v \in cand_{O,i} \mid \exists x \in O : (x, v) \in E_i\}$
9: $u := \arg\max_{v \in neighbors} \{\sum_{i \in S_O} deg^O_{G_i}(v)\}$
10: EXPAND$(O, u, S_O, \{cand_{O,i} \mid i \in S_O\})$
11: **return** Result
12: **procedure** EXPAND$(O, u, S_O, \{cand_{O,i} \mid i \in S_O\})$
13: $O_{next} := O \cup \{u\}$, $S_{O_{next}} := \{i \in S_O \mid u \in cand_{O,i}\}$
14: **for all** $i \in S_{O_{next}}$ **do** $cand_{O_{next},i} := cand_{O,i} \setminus \{u\}$
15: Prune $S_{O_{next}}$ and $cand_{O_{next},i}$ $(\forall i \in S_{O_{next}})$
16: $ST_{next} = (O_{next}, S_{O_{next}}, \{cand_{O_{next},i} \mid i \in S_{O_{next}}\})$
17: **if** $Q_{est}(ST_{next}) \geq 0$ **then** $queue.insert(ST_{next})$
18: **for all** $i \in S_O$ **do** $cand_{O,i} := cand_{O,i} \setminus \{u\}$
19: Prune S_O and $cand_{O,i}$ $(\forall i \in S_O)$
20: $ST_{remain} = (O, S_O, \{cand_{O,i} \mid i \in S_O\})$
21: **if** $Q_{est}(ST_{remain}) \geq 0$ **then** $queue.insert(ST_{remain})$
22: **if** $\exists$ cluster $C = (O_{next}, S)$, $S \subseteq S_{O_{next}}$ **then**
23: **if** $\neg\exists C' \in Result : C \prec_{red} C'$ **then** $queue.insert(C)$

Πηγή: Boden et al (2012)

5.2 Εκτέλεση του αλγόριθμου MiMAG

Δοσμένου ενός multi-layer γραφήματος G, ο MiMAG υπολογίζει ένα αποτέλεσμα με τη βέλτιστη ομαδοποίηση χωρίς πλεονάζοντα cluster. Αρχικά, το σύνολο Result είναι άδειο (σειρά 1) και γεμίζει διαδοχικά κατά την εκτέλεση. Όταν αρχίζει η εκτέλεση η ουρά περιέχει ένα στοιχείο, το οποίο αναπαριστά τον αρχικό κόμβο (ρίζα - root) του extended set enumeration tree (σειρά 2). Όσο η ουρά περιέχει στοιχεία, αυτό με την μεγαλύτερη (εκτιμώμενη) ποιότητα αφαιρείται από την ουρά. Αν το αντικείμενο αυτό είναι cluster τότε δεν μπορεί να βρεθεί κάποιο με καλύτερη ποιότητα, άρα το προσθέτουμε στο σύνολο του Result, υπό την προϋπόθεση ότι δεν είναι πλεονάζον σε σχέση με κάποιο άλλο cluster που ανήκει στο σύνολο του Result (σειρά 6). Αν το αντικείμενο είναι υποδέντρο, τότε επεκτείνουμε το αντιπροσωπευτικό του σύνολο κορυφών O κατά μία γειτονική κορυφή u, που περιέχεται στο σύνολο υποψηφίων κορυφών για επέκταση $cand_{O,i}$. Χρησιμοποιούμε αυτή με το μεγαλύτερο βαθμό, γιατί θα έχει τη μεγαλύτερη πιθανότητα να μας οδηγήσει σε πιο πυκνά γραφήματα.

Εικόνα 32 - Σχήμα επέκτασης ενός κόμβου



Πηγή: Ιδία επεξεργασία

Η επέκταση ενός κόμβου O, ξεκινάει όταν ο MiMAG καλεί την συνάρτηση EXPAND για το υποδέντρο με σύνολο κορυφών O. Σε αυτήν την συνάρτηση αρχικά καθορίζονται το σύνολο κορυφών O_{next}, το σύνολο των ενεργών layer S_{Onext} και το σύνολο υποψηφίων $cand_{Onext,i}$. Το S_{Onext} μπορεί να περιέχει τα layer i στα οποία η κορυφή προς επέκταση u υπήρχε στο αντίστοιχο $cand_{Onext,i}$. Μόλις το STnext δημιουργηθεί, προσθέτεται στην ουρά εάν η ποιότητα του δεν είναι αρνητική. Το ίδιο συμβαίνει και για το STremain, όμως να διευκρινιστεί πως υπολογίζεται εκ νέου η ποιότητα του υποδέντρου, αφού έχει αφαιρεθεί η κορυφή u από το σύνολο των υποψηφίων $cand_{O,i}$. Τέλος, ελέγχεται αν το cluster με σύνολο κορυφών Onext είναι έγκυρο και όχι πλεονάζον και εφόσον είναι, το προσθέτουμε επίσης στην ουρά.

5.3 Ποιότητα υποδέντρων

Παρακάτω, παρουσιάζονται τα άνω όρια για την ποιότητα των υποδέντρων. Αρχικά, πρέπει να εκμεταλλευτούμε το γεγονός πως κάποια υποδέντρα δεν περιέχουν κανένα ενδιαφέρον cluster. Σε αυτές τις περιπτώσεις το άνω όριο της ποιότητας αυτών των υποδέντρων πρέπει να οριστεί ως -1, έτσι ώστε να μην προστεθούν στην ουρά.

Η πρώτη περίπτωση εύρεσης υποδέντρων που δεν περιέχουν ενδιαφέρον cluster χρησιμοποιεί τα ενεργά layer που έχουν απομείνει στο σύνολο κορυφών O. Αν δεν έχουν απομείνει ενεργά layer μπορούμε με βεβαιότητα να συμπεράνουμε πως δεν γίνεται να βρεθεί κάποιο έγκυρο cluster σε κάποιο υποδέντρο με ρίζα το O.

Στην δεύτερη περίπτωση θα χρησιμοποιήσουμε το μοντέλο πλεονασμού για να καθορίσουμε το όριο. Αν όλα τα cluster C που υπάρχουν σε ένα υποδέντρο ST (για παράδειγμα C = (X, S_X) με $S_X \subseteq S_O$ $O \subset X \subseteq O$ υ $\cup_i \in s_o$ $cand_{O,i}$) είναι πλεονάζοντα σε σχέση με κάποιο ή κάποια cluster C’ ∈ Result τότε το υποδέντρο δεν μπορεί να προστεθεί στην ουρά, γιατί δεν θα περιέχει κανένα ενδιαφέρον cluster. Για αυτό τον λόγο, ακόμα και αν η ποιότητα του ST είναι θετική, μπορούμε να εκτιμήσουμε την ποιότητα του ως -1. Για να ελέγξουμε αν ένα ST είναι πλεονάζον (δηλαδή περιέχει μόνο πλεονάζοντα cluster) πρέπει να ελέγξουμε τις ιδιότητες του ορισμού 4. Η πρώτη ιδιότητα C ≠ C’ και η δεύτερη Q(C) ≤ Q(C’) εκπληρώνεται για κάθε πιθανό C. Μόνο η τρίτη ιδιότητα, για την επικάλυψη των ακμών, πρέπει να ελεγχθεί ($\frac{1}{|S_X|}\sum_{i \in S_X \cap S'} \frac{|E_i(X) \cap E_i(O')|}{|E_i(X)|} \geq r$).

Έτσι λοιπόν, καθορίζουμε ένα κατώτερο όριο oul_{min} για την επικάλυψη ακμών (για όλα τα πιθανά cluster C) ώστε:

$$oul_{min} \leq \frac{1}{|S_X|}\sum_{i \in S_X \cap S'} \frac{|E_i(X) \cap E_i(O')|}{|E_i(X)|}$$

Έπειτα, εάν $oul_{min} \geq r$ θέτουμε το Qest(ST) = -1 γιατί υπάρχει επικάλυψη σε τέτοιο βαθμό ώστε να θεωρείται πλεονάζον. Για κάθε υποδέντρο ST και για κάθε cluster C’ = (O’, S’) ∈ Result με S’ ⊇ So έχουμε:

$$oul_{min} = min_{i \in So} \frac{|E_i(O \cap O')| + \max\{\frac{1}{4}\cdot(|O|^2 + |O|) - |E_i(O \cap O')| - k,\ 0\}}{|E_i(O \cap O')| + k + \max\{\frac{1}{4}\cdot(|O|^2 + |O|) - |E_i(O \cap O')| - k,\ 0\}}$$

όπου $k = |\ E_i\ (O \cup cand_{O,i}) \setminus E_i\ ((O \cup cand_{O,i}) \cap O')\ |$

Χρήσιμες ιδιότητες στις συναρτήσεις ποιότητας των cluster είναι της πυκνότητα και της πληθικότητας (cardinality). Θέτουμε λοιπόν, κάποια άνω όρια για αυτές τις ιδιότητες. Δεδομένου ενός υποδέντρου ST = {O, So, $cand_{O,i}$ | i ∈ So}, για κάθε μονοδιάστατο MLCS cluster X στο layer i ∈ So με O ⊂ X ⊆ O ∪ $cand_{O,i}$:

- $\gamma(X) \leq \min \{\frac{\min\_deg_{G_i}}{|O|}, 1\} =: \gamma_i^{max}$ with
  $\min\_deg_{Gi} = min_{v \in O} \{deg^{O \cup cand_{O,i}}(v)\}$
- $|X| \leq \min (\left\lfloor \frac{\min\_deg_{G_i}}{0,5} \right\rfloor + 1, |O \cup cand_{O,i}|) =: n_i^{max}$
- $|E_i(X)| \leq |E_i(O)| + (n_i^{max} - |O|) \cdot max_{v \in cand_O}\{deg_{Gi}^{O \cup cand_{O,i}}(v)\}$

Μπορούμε να χρησιμοποιήσουμε τα παραπάνω άνω όρια για να συγκεκριμενοποιήσουμε τη συνάρτηση ποιότητας των υποδέντρων. Η ποιότητα ενός υποδέντρου ST έχει ώς άνω όριο

$$Qest(ST) = max_{k \in \{1,\dots,|So|\}}\{max_k(n_i^{max}) \cdot \sum_{m=1}^{k} max_m(\gamma_i^{max})\}$$

όπου το $max_x(y_i)$ υποδηλώνει την x-οστή, μεγαλύτερη τιμή, από όλα τα $\{y_i \mid i \in So\}$. Επιπλέον, αν έχουμε $max_{i \in So}(n_i^{max}) < 8$ ή $|So| < 2$ το υποδέντρο δεν γίνεται να περιέχει κάποιο cluster με θετική ποιότητα και η εκτίμηση του Qest(ST) = -1.

6. Τεχνικές κλαδέματος και βελτίωσης

6.1 Τεχνικές κλαδέματος

Για την βελτίωση της απόδοσης του αλγορίθμου θα χρησιμοποιήσουμε κάποιες τεχνικές κλαδέματος του extended set enumeration tree. Κλαδεύοντας το, μειώνουμε το εύρος αναζήτησης σημαντικά, βελτιώνοντας τον χρόνο εύρεσης των κοινοτήτων. Σκοπός μας είναι να αφαιρέσουμε από τη λίστα υποψηφίων $cand_{O,i}$ όσες κορυφές δεν θα μπορούν να δημιουργήσουν κάποιο αποδεκτό cluster ή να απορρίψουμε υποδέντρα τα οποία δεν περιέχουν κανένα ενδιαφέρον cluster, δηλαδή να μην τα προσθέσουμε στην ουρά.

Μέθοδος 1:

Η πρώτη μέθοδος είναι ένας απλός έλεγχος αν το υποδέντρο ως προς επέκταση (STnext) που δημιουργείται, είναι όμοιο με κάποιο που βρίσκεται ήδη στην ουρά. Σε αυτήν την περίπτωση, απορρίπτουμε το καινούργιο υποδέντρο και δεν το προσθέτουμε στην ουρά, γιατί ένα διπλότυπο δεν θα μας δώσει κάποιο επιπλέον αποτέλεσμα.

Μέθοδος 2:

Η δεύτερη μέθοδος ασχολείται με την διαγραφή κορυφών από τη λίστα υποψηφίων για επέκταση ενός υποδέντρου ($cand_{O,i}$). Εάν κάποια κορυφή ανήκει σε λιγότερα layer από αυτά που χρειάζεται για να είναι αποδεκτό ένα cluster, τότε μπορούμε να το διαγράψουμε με ασφάλεια από τη λίστα, αφού ποτέ δεν πρόκειται να μας οδηγήσουν σε κάποιο cluster το οποίο να έχει ποιότητα μεγαλύτερη από -1.

Μέθοδος 3[7][11]:

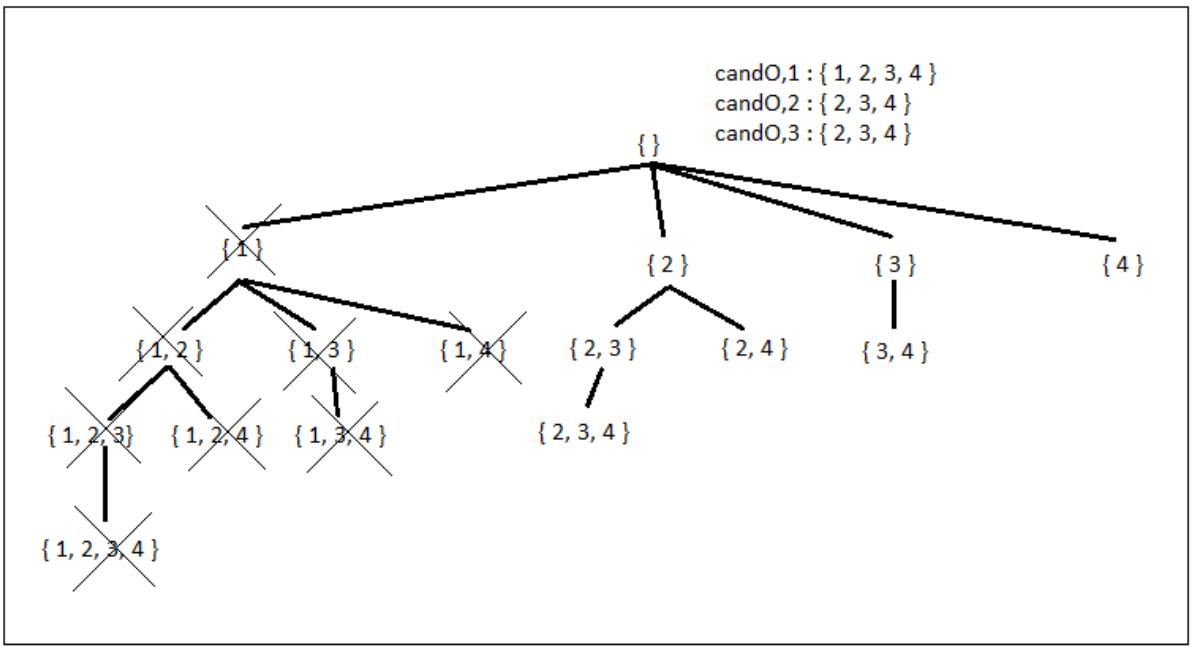
Σε αυτή τη μέθοδο θα αναζητήσουμε τις κορυφές οι οποίες έχουν βαθμό μικρότερο από $\lceil \gamma \cdot (\min\_size - 1) \rceil$ σε κάποιο υποδέντρο. Το min_size είναι ο ελάχιστος αριθμός κορυφών που απαιτούμε να περιέχει ένα cluster. Ο βαθμός μπορεί να υπολογιστεί από το υπογράφημα που σχηματίζεται από την λίστα κορυφών O και την λίστα υποψηφίων $cand_{O,i}$ ($O \cup cand_{O,i}$). Αν κάποια κορυφή έχει μικρότερο βαθμό, μπορεί να αφαιρεθεί από την λίστα υποψηφίων $cand_{O,i}$ διότι καμιά αποδεκτή γ-quasi-clique δεν μπορεί να την περιέχει.

Μέθοδος 4[7][11]:

Στην μέθοδο 4, που είναι και η τελευταία που υλοποιήθηκε στα πλαίσια αυτής της εργασίας ελέγχουμε αν στο σύνολο κορυφών O ενός υποδέντρου, περιέχεται κάποια κορυφή η οποία έχει βαθμό μικρότερο από $\lceil \gamma \cdot (\min\_size - 1) \rceil$. Ομοίως με πριν, ο βαθμός μπορεί να υπολογιστεί από το υπογράφημα που σχηματίζεται από την λίστα κορυφών O και την λίστα υποψηφίων $cand_{O,i}$ ($O \cup cand_{O,i}$). Αν υπάρχει λοιπόν, κάποια κορυφή με μικρότερο βαθμό, τότε ολόκληρο το υποδέντρο μπορεί να κλαδευτεί γιατί πλέον δεν είναι έγκυρο. Αυτό μπορεί να προκύψει καθώς διαγράφονται κορυφές από την λίστα υποψηφίων $cand_{O,i}$.

Οι μέθοδοι κλαδέματος επαναλαμβάνονται διαδοχικά μέχρι να μην μπορεί να διαγραφεί κάποια περαιτέρω κορυφή. Ο λόγος που επαναλαμβάνονται είναι γιατί αν διαγραφεί κάποια κορυφή, τότε ο βαθμός κάποιας άλλης μπορεί να αλλάξει ή μπορεί μια κορυφή πλέον να βρίσκεται σε λιγότερα layer στη λίστα υποψηφίων $cand_{O,i}$ από όσα είναι απαραίτητο. Καθώς διαγράφονται κορυφές από την λίστα, υπάρχει περίπτωση κάποια κορυφή που ανήκει στο σύνολο κορυφών O να καταλήξει να έχει βαθμό μικρότερο του αποδεκτού και συνεπώς, το υποδέντρο πρέπει να κλαδευτεί.

Εικόνα 33 - Παράδειγμα διαγραφής υποδέντρου που ανήκει σε 1 layer



Πηγή: Ιδία επεξεργασία

6.2 Τεχνικές βελτίωσης

Βελτίωση 1:

Ο αλγόριθμος βρίσκει τα cluster με αύξουσα σειρά. Για παράδειγμα, πρώτα θα βρει το cluster που έχει 8 κορυφές και ύστερα θα επεκταθεί και θα βρει το cluster 9 που έχει κορυφές. Έτσι, στην λίστα του αποτελέσματος τα μικρότερα cluster παραμένουν και έχουν γίνει πλεονάζοντα. Όσα cluster υπάρχουν στην λίστα αποτελεσμάτων χρησιμοποιούνται για συγκρίσεις όταν δημιουργείται ένα καινούργιο cluster για να βρεθεί αν είναι πλεονάζον. Η παραμονή τους στην λίστα αποτελεσμάτων οδηγεί σε ένα πολύ μεγάλο αριθμό συγκρίσεων που δεν χρειάζεται. Το πρόβλημα λύνεται αν κάθε φορά που προσθέτουμε ένα cluster στη λίστα αποτελεσμάτων, την αδειάσουμε και αρχίσουμε να τη γεμίζουμε με τα στοιχεία ανάλογα με την μεγαλύτερη ποιότητα, χρησιμοποιώντας έναν προσωρινό buffer και ελέγχοντας κάθε φορά για πλεονασμούς.

Βελτίωση 2:

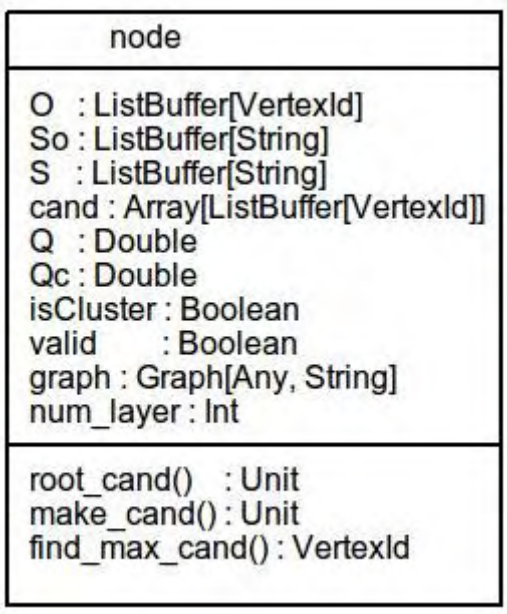
Επειδή ο αλγόριθμος εξετάζει τους γείτονες κάθε φορά για την επέκταση ενός υποδέντρου, οδηγείται αρκετές φορές σε υποδέντρα με μεγάλο πλήθος κορυφών τα οποία έχουν πάρα πολύ μικρή πυκνότητα και δεν υπάρχει καμιά περίπτωση να βρεθεί αποδεκτό cluster σε αυτά. Ένα πολύ μεγάλο μέρος χρόνου και μνήμης σπαταλιέται σε αυτά τα υποδέντρα χωρίς κάποιο ουσιαστικό αποτέλεσμα. Για αυτό το λόγο, η βελτίωση που έγινε κλαδεύει του κόμβους που δεν δημιουργούν cluster όταν περιέχει 8 κορυφές. Αντίθετα, αν ένας κόμβος δημιουργεί κάποιο cluster είναι πιθανό να δημιουργεί και κάποιος απόγονος του, οπότε δεν κλαδεύεται. Η υλοποίηση αυτή μπορεί όμως, να μας οδηγήσει στην παράβλεψη κάποιων αποδεκτών cluster. Αυτό μπορεί να επιλυθεί. Αν το τρέξιμο του λογισμικού φτάσει αίσια στο τέλος του, τότε μπορούμε να αλλάξουμε τις παραμέτρους αναζήτησης ώστε να γίνεται το κλάδεμα από 9 ή περισσότερες κορυφές.

7. Προγραμματιστικό μέρος - Έξοδος προγράμματος

7.1 Προγραμματιστικό μέρος

Στο προγραμματιστικό κομμάτι, ο κάθε κόμβος αναπαρίσταται από ένα αντικείμενο κλάσης node.

Εικόνα 34 - Κλάση node



Πηγή: Ιδία επεξεργασία

Επεξήγηση συναρτήσεων της κλάσης node:

- *root_cand()* : Φτιάχνει την λίστα των υποψήφιων κορυφών ως προς επέκταση για τον αρχικό κόμβο (ρίζα). Η συνάρτηση δεν επιστρέφει τίποτα.
- *make_cand()* : Επεκτείνει την λίστα των υποψηφίων ανάλογα με την κορυφή που μόλις προστέθηκε. Η συνάρτηση δεν επιστρέφει τίποτα.
- *find_max_cand()* : Εντοπίζει την κορυφή που έχει το μεγαλύτερο βαθμό, συγκριτικά με όλες τις υπόλοιπες κορυφές, σε όλα τα layer. Η συνάρτηση επιστρέφει την κορυφή αυτή, τύπου VertexId.

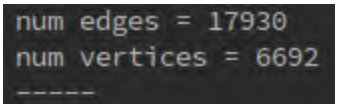
Επεξήγηση βασικών συναρτήσεων του λογισμικού:

- *checkifRedundant()* : Συγκρίνει δύο cluster για να αποφανθεί αν υπάρχει κάποιο πλεονάζον. Επιστρέφει μια Boolean τιμή.
- *expand()* : Η συνάρτηση αυτή είναι υπεύθυνη για την επέκταση ενός κόμβου. Δημιουργεί τον νέο κόμβο STnext ο οποίος αν δεν κλαδευτεί θα προστεθεί στην ουρά. Επίσης, ελέγχει για ενδιαφέροντα cluster και επεξεργάζεται τον κόμβο-γονιό STremain.
- *calc_quality()* : Αποφασίζει αν είναι «ενδιαφέρον» ένα cluster και υπολογίζει την τιμή της ποιότητας του.
- *calc_Qest()* : Αποφασίζει αν περιέχει «ενδιαφέρον» cluster ένα υποδέντρο και υπολογίζει την τιμή της ποιότητας του.
- *calc_k()* : Υπολογίζει την τιμή k που χρησιμοποιείται για τον υπολογισμό της ποιότητας των υποδέντρων. Επιστρέφει μια τιμή σε Int.
- *calc_ouli()* : Υπολογίζει την τιμή oul_{min} που χρησιμοποιείται για τον υπολογισμό της ποιότητας των υποδέντρων. Επιστρέφει μια τιμή σε Double.
- *find_min_degGi()* : Υπολογίζει την τιμή min_deg_{Gi} που χρησιμοποιείται για τον υπολογισμό της ποιότητας των υποδέντρων. Επιστρέφει μια τιμή σε Double.
- *find_nimax()* : Υπολογίζει την τιμή n_i^{max} που χρησιμοποιείται για τον υπολογισμό της ποιότητας των υποδέντρων. Επιστρέφει μια τιμή σε Double.
- *prune1()* : Εκτελεί την μέθοδο κλαδέματος 2.
- *prune2()* : Εκτελεί μέρος της μεθόδου κλαδέματος 3.
- *prune_ceil()* : Εκτελεί μέρος της μεθόδου κλαδέματος 3.

7.2 Έξοδος προγράμματος

Σε αυτό το σημείο θα αναλυθεί η έξοδος του λογισμικού κατά τη διάρκεια εκτέλεσης. Ξεκινώντας λοιπόν, υπολογίζονται οι ακμές και οι κορυφές που περιέχει το δίκτυο που αναλύουμε, αφαιρώντας τα διπλότυπα αν υπάρχουν:

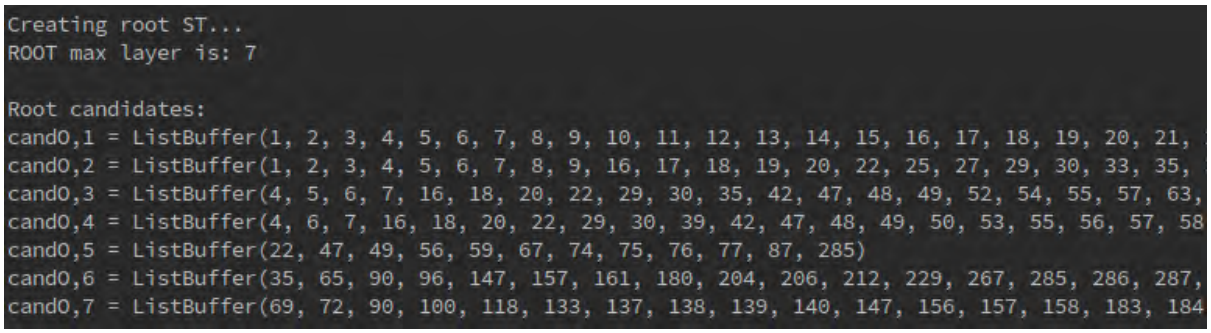
Εικόνα 35 - Έξοδος 1



Πηγή: Ιδία επεξεργασία

Έπειτα, δημιουργείται ο κόμβος ρίζας (ROOT):

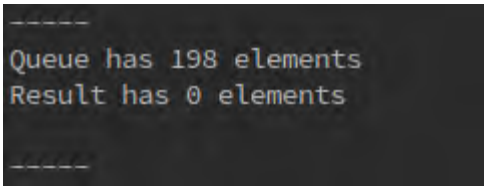
Εικόνα 36 - Έξοδος 2



Πηγή: Ιδία επεξεργασία

Πριν από κάθε βήμα της επανάληψης, για να υπάρχει μια εικόνα του σημείου που βρισκόμαστε, εμφανίζονται οι αριθμοί των στοιχείων που περιέχει η ουρά και το σύνολο αποτελεσμάτων:

Εικόνα 37 - Έξοδος 3



Πηγή: Ιδία επεξεργασία

Κάθε βήμα, μέσα στην επανάληψη είναι αριθμημένο. Αναφέρεται το αν είναι υποδέντρο ή cluster και συνεχίζει ανάλογα:

Εικόνα 38 - Έξοδος 4

```
Step 853: Current node is: List(180, 659, 660, 670, 678, 717, 716, 1698, 1708) with Q = 12.375
Sum of queue elements with quality 12.375 is 0
Current node is cluster!
***Cluster added to Result***
Checking if there are redundant clusters in result...
New result contains:
Cluster with O: ListBuffer(180, 659, 660, 715, 1698, 670, 717, 716, 1708) S: ListBuffer(1, 2) Q: 12.375
Cluster with O: ListBuffer(180, 659, 660, 670, 678, 717, 716, 1698, 1708) S: ListBuffer(1, 2) Q: 12.375
```

Πηγή: Ιδία επεξεργασία

Στην περίπτωση που είναι cluster γίνονται κάποιοι έλεγχοι για το cluster και μπορεί να προστεθεί στο σύνολο αποτελεσμάτων αν όλα είναι εντάξει.

Εικόνα 39 - Έξοδος 5

```
Step 0: Current node is: List() with Q = 0.0
Current node is subtree!
Max Cand Vertex is: 180 (180,145)
Expanding ListBuffer() subtree...
```

Πηγή: Ιδία επεξεργασία

Στην περίπτωση που είναι υποδέντρο, γίνεται η επέκταση σε STnext και STremain και ελέγχεται αν υπάρχει αποδεκτό cluster με τις κορυφές που περιέχει το υποδέντρο STnext, αν το STnext δημιουργηθεί και έχει αποδεκτή ποιότητα.

Εικόνα 40 - Έξοδος 6

```
->STnext cluster/subtree created with .O = ListBuffer(180, 672) .So = ListBuffer(1, 2)
before prune:
candO,1: ListBuffer(663, 665, 669, 670, 716, 717, 985, 1708, 462, 659, 660, 235, 680, 684, 689, 714, 812, 909, 1688, 1698, 17
candO,2: ListBuffer(663, 665, 669, 670, 716, 717, 985, 1708, 683, 886, 1366, 1837, 1871, 1873, 1874, 1898, 1948, 1949, 1950)
after prune:
candO,1: ListBuffer(663, 665, 669, 670, 716, 717, 985, 1708)
candO,2: ListBuffer(663, 665, 669, 670, 716, 717, 985, 1708)
pruned from cand: 34
Calculating Qest...
STnext Qest():20.0
```

Πηγή: Ιδία επεξεργασία

Εικόνα 41 - Έξοδος 7

```
Checking if cluster C = (Onext, S) exists... , where S c So
Onext = ListBuffer(1646) S = ListBuffer(1, 2)
***Cluster not created***
```

Πηγή: Ιδία επεξεργασία

Εικόνα 42 - Έξοδος 8

```
Checking if cluster C = (Onext, S) exists... , where S c So
Onext = ListBuffer(180, 659, 660, 715, 672, 670, 678, 663) S = ListBuffer(1, 2)
Q = 9.142857142857142
***Cluster added in queue!***
```

Πηγή: Ιδία επεξεργασία

Εικόνα 43 - Έξοδος 9

```
->STremain cluster/subtree created with .O = ListBuffer() .So = ListBuffer(1, 2, 3, 4, 5, 6, 7)
before prune:
candO,1: ListBuffer(48, 74, 111, 167, 222, 276, 287, 311, 333, 375, 381, 391, 397, 399, 400, 475, 489, 570, 571, 573, 702, 712, 722,
candO,2: ListBuffer(47, 48, 111, 167, 212, 222, 276, 278, 296, 311, 329, 333, 375, 378, 380, 391, 397, 399, 400, 475, 489, 570, 571,
candO,3: ListBuffer(121, 1632)
candO,4: ListBuffer(121, 1632, 1678, 1679, 1683, 1684, 1685)
candO,5: ListBuffer(47, 74)
candO,6: ListBuffer(212, 267, 287, 296, 329, 996, 1267, 1506)
candO,7: ListBuffer(267, 278, 287, 378, 380, 381, 729, 984, 1128, 1265, 1324, 1399, 2067, 2537, 2542, 3261, 3711, 5087, 6002, 6060)
after prune:
candO,1: ListBuffer(48, 74, 111, 167, 222, 276, 287, 311, 333, 375, 381, 391, 397, 399, 400, 475, 489, 570, 571, 573, 702, 712, 722,
candO,2: ListBuffer(47, 48, 111, 167, 212, 222, 276, 278, 296, 311, 329, 333, 375, 378, 380, 391, 397, 399, 400, 475, 489, 570, 571,
candO,3: ListBuffer(121, 1632)
candO,4: ListBuffer(121, 1632, 1678, 1679, 1683, 1684, 1685)
candO,5: ListBuffer(47, 74)
candO,6: ListBuffer(212, 267, 287, 296, 329, 996, 1267, 1506)
candO,7: ListBuffer(267, 278, 287, 378, 380, 381, 729, 984, 1128, 1265, 1324, 1399, 2067, 2537, 2542, 3261, 3711, 5087, 6002, 6060)
pruned from cand: 0
Calculating Qest...
STremain Qest():1708.0
```

Πηγή: Ιδία επεξεργασία

Μετά από τη δημιουργία είτε και του STnext και του STremain ακολουθεί ένα μήνυμα το οποίο μας ενημερώνει για το αν το υποδέντρο προστέθηκε στην ουρά ή όχι ή αν τελείωσαν οι υποψήφιοι ως προς επέκταση του υποδέντρου:

Εικόνα 44 - Έξοδος 10

```
***Subtree added to queue***
```

Πηγή: Ιδία επεξεργασία

Εικόνα 45 - Έξοδος 11

```
***Subtree end. No more candidates***
```

Πηγή: Ιδία επεξεργασία

Εάν ένα cluster είναι πλεονάζον τότε εμφανίζεται το ακόλουθο μήνυμα:

Εικόνα 46 - Έξοδος 12

```
Checking if cluster C = (Onext, S) exists... , where S c So
Onext = ListBuffer(180, 660, 715, 659, 670, 663, 717, 665) S = ListBuffer(1, 2)
Q = 9.142857142857142
Deleted from |S|: ListBuffer()
1/S * (iEi / Ei(O)) = 0.6333333333333333 >= r = 0.25
Cluster is redundant!

***Cluster is redundant!***
```

Πηγή: Ιδία επεξεργασία

Όταν φτάσει στο τέλος το πρόγραμμα εμφανίζεται το σύνολο των αποτελεσμάτων και ο χρόνος εκτέλεσης:

Εικόνα 47 - Έξοδος 13

```
-----
Queue has 0 elements
Result has 3 elements

-----

RESULT =
1) Cluster: ListBuffer(180, 659, 660, 670, 715, 716, 717, 1698, 1708)    S: ListBuffer(1, 2)     Q: 12.375
2) Cluster: ListBuffer(180, 659, 660, 670, 678, 716, 717, 1698, 1708)    S: ListBuffer(1, 2)     Q: 12.375
3) Cluster: ListBuffer(180, 660, 663, 665, 669, 670, 672, 985)   S: ListBuffer(1, 2)     Q: 10.285714285714285

Elapsed time: 0 hours 17 minutes 37 seconds


Process finished with exit code 0
```

Πηγή: Ιδία επεξεργασία

8. Πειραματικό μέρος

8.1 Περιγραφή συστήματος και προδιαγραφών

Στο πειραματικό κομμάτι, όπου εξετάζεται η ορθότητα, η απόδοση και οι δυνατότητες του λογισμικού εύρεσης κοινοτήτων, χρησιμοποιήθηκαν πραγματικά δίκτυα. Τα δίκτυα αυτά είναι μη κατευθυνόμενα, χωρίς βάρη και είναι multiplex. Στην περίπτωση που είναι κατευθυνόμενα ή έχουν βάρη, στα συγκεκριμένα πειράματα δεν το λαμβάνουμε υπόψη.

Τα ενδιαφέροντα cluster που αναζητούνται ως κοινότητες, περιέχουν τουλάχιστον 8 κορυφές, πρέπει να ανήκουν σε τουλάχιστον 2 layer και να έχουν πυκνότητα, όπως έχει οριστεί παραπάνω, $\gamma \geq 0{,}5$. Επίσης, τα cluster τα οποία είναι πλεονάζοντα δεν συμπεριλαμβάνονται στη λίστα των αποτελεσμάτων.

Η αναπαράσταση των δικτύων έχει γίνει χρησιμοποιώντας το λογισμικό Gephi και το Pajek. Η διαδικασία για την αναπαράσταση περιλαμβάνει τον διαχωρισμό των ακμών του κάθε layer σαν ξεχωριστό γράφημα, διότι τα λογισμικά δεν υποστηρίζουν πολυεπίπεδα δίκτυα.

Τα δεδομένα των δικτύων βρίσκονται: http://deim.urv.cat/~manlio.dedomenico/data.php .
Τα πειράματα έχουν γίνει σε λειτουργικό σύστημα Ubuntu Linux 16.04 (x64) με Intel Core i7-45000 CPU @ 1.80GHz 2,4GHz.

8.2 1^{ο} Δίκτυο

“Arabidopsis_largest.txt”[13][14]**:**

Μελετιούνται διαφορετικοί τύποι γενετικών αλληλεπιδράσεων για οργανισμούς στο Biological General Repository for Interaction Datasets (BioGRID, thebiogrid.org), μια δημόσια βάση δεδομένων που αρχειοθετεί δεδομένα από γενετικές αλληλεπιδράσεις και αλληλεπιδράσεις πρωτεϊνών από ανθρώπινα οργανισμούς και από μοντέλα οργανισμών.

Το multiplex δίκτυο χρησιμοποιεί τα παρακάτω layer:

1. Άμεση επαφή
2. Σωματικό συσχετισμό
3. Προσθετική γενετική αλληλεπίδραση καθορισμένη από ανισότητα
4. Κατασταλτική γενετική αλληλεπίδραση καθορισμένη από ανισότητα
5. Συνθετική γενετική αλληλεπίδραση καθορισμένη από ανισότητα
6. Συσχετισμό
7. Colocalization

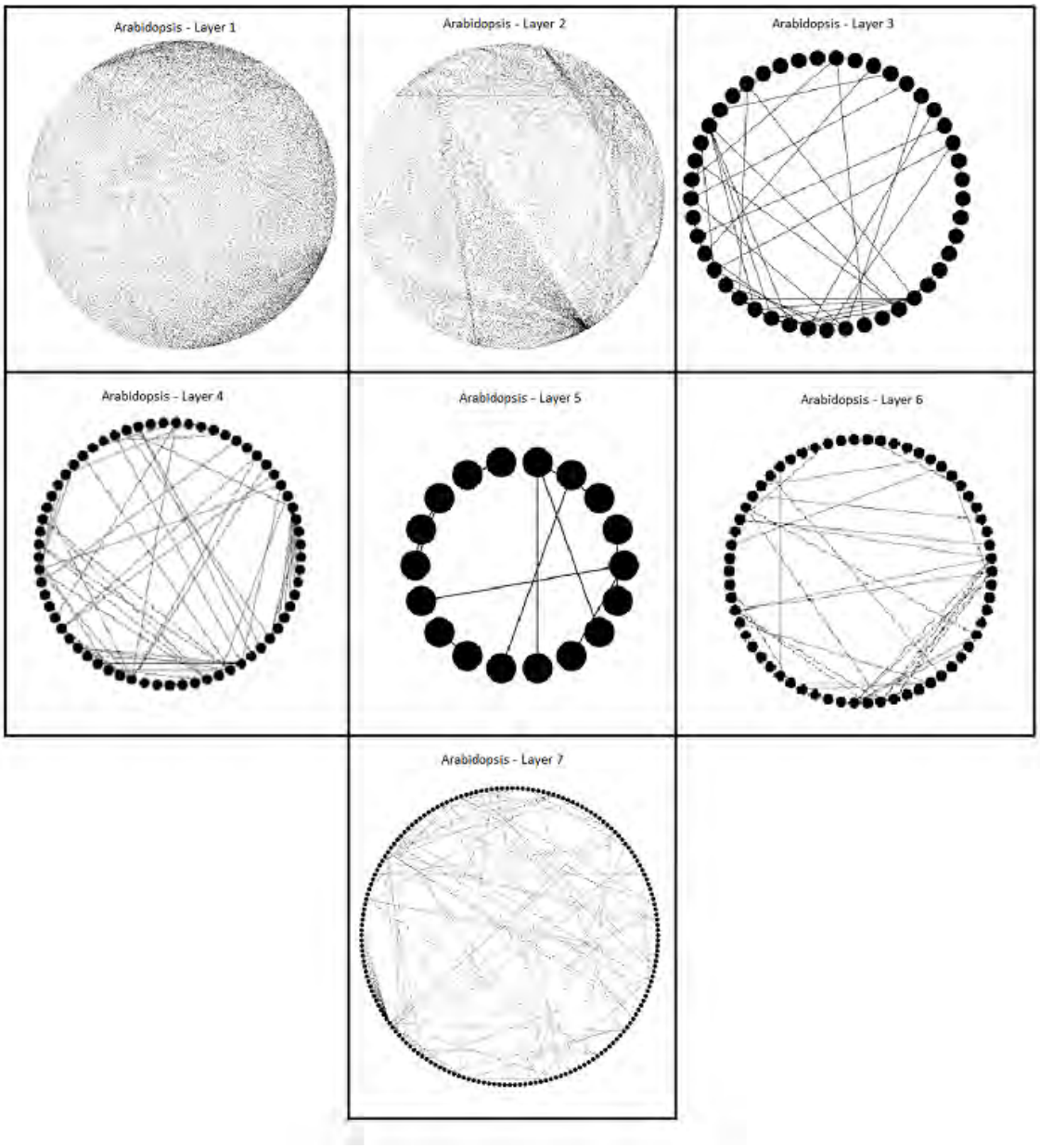
Αριθμός ακμών: 17.930

Αριθμός κορυφών: 6.692

Πλήθος layer: 7

Παρακάτω υπάρχει η απεικόνιση του δικτύου:

Εικόνα 48 - Απεικόνιση του δικτύου Arabidopsis

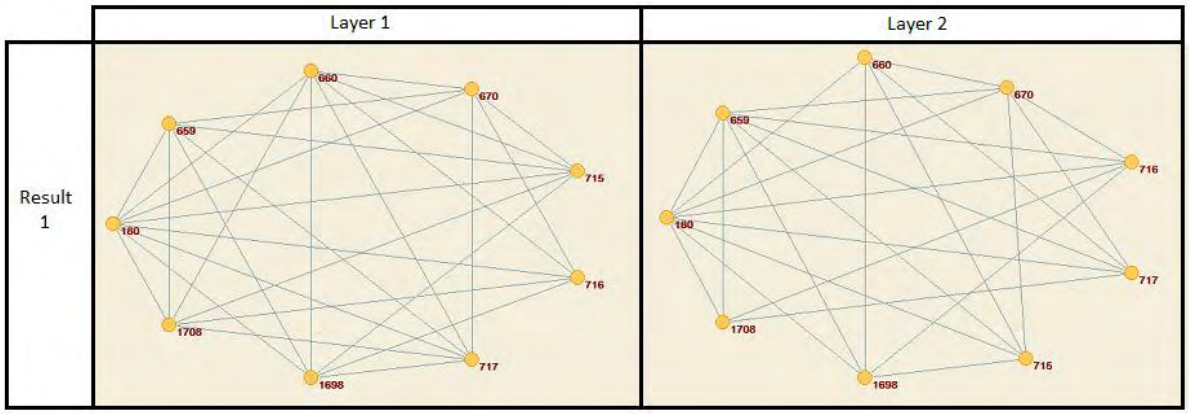


Πηγή: Ιδία επεξεργασία

Τα αποτελέσματα του δικτύου είναι οι κοινότητες:

1) Cluster: (180, 659, 660, 715, 1698, 670, 717, 716, 1708) Layers: (1, 2) Quality: 12.375
2) Cluster: (180, 659, 660, 715, 670, 678, 717, 665, 669) Layers: (1, 2) Quality: 12.375
3) Cluster: (180, 659, 672, 670, 715, 1698, 717, 660, 1708) Layers: (1, 2) Quality: 12.375
4) Cluster: (180, 659, 660, 715, 672, 670, 716, 1698, 1708) Layers: (1, 2) Quality: 12.375
5) Cluster: (180, 659, 660, 670, 678, 717, 716, 1698, 1708) Layers: (1, 2) Quality: 12.375
6) Cluster: (180, 659, 660, 715, 672, 670, 678, 717, 1708) Layers: (1, 2) Quality: 12.375
7) Cluster: (180, 659, 660, 715, 670, 678, 717, 716, 1708) Layers: (1, 2) Quality: 12.375
8) Cluster: (180, 659, 660, 672, 670, 717, 716, 1698, 1708) Layers: (1, 2) Quality: 12.375
9) Cluster: (180, 659, 660, 672, 670, 678, 717, 716, 1708) Layers: (1, 2) Quality: 12.375
10) Cluster: (180, 659, 660, 672, 670, 678, 716, 1698, 1708) Layers: (1, 2) Quality: 12.375
11) Cluster: (180, 659, 660, 715, 672, 670, 678, 716, 1708) Layers: (1, 2) Quality: 12.375
12) Cluster: (180, 660, 672, 670, 715, 1698, 717, 716, 1708) Layers: (1, 2) Quality: 12.375

Εικόνα 49 - Παραδειγμα αποτελέσματος 1



Πηγή: Ιδία επεξεργασία

8.3 2^{ο} Δίκτυο

“Auger_largest.txt”[15]**:**

Το δίκτυο αποτελείται από layer που αφορούν διαφορετικά καθήκοντα μέσα στο Pierre Auger Collaboration.

Το multiplex δίκτυο χρησιμοποιεί τα παρακάτω layer:

1. Νετρόνια
2. Ανιχνευτής
3. Βελτιώσεις
4. Ανισοτροπία
5. Σημείο-Πηγή
6. Σύσταση μάζας
7. Οριζόντια
8. Υβριδική ανακατασκευή
9. Φάσμα
10. Φωτόνια
11. Ατμοσφαιρικός
12. SD – ανακατασκευή
13. Αδρονικές αλληλεπιδράσεις
14. Exotics
15. Μαγνητικό
16. Αστροφυσικά σενάρια

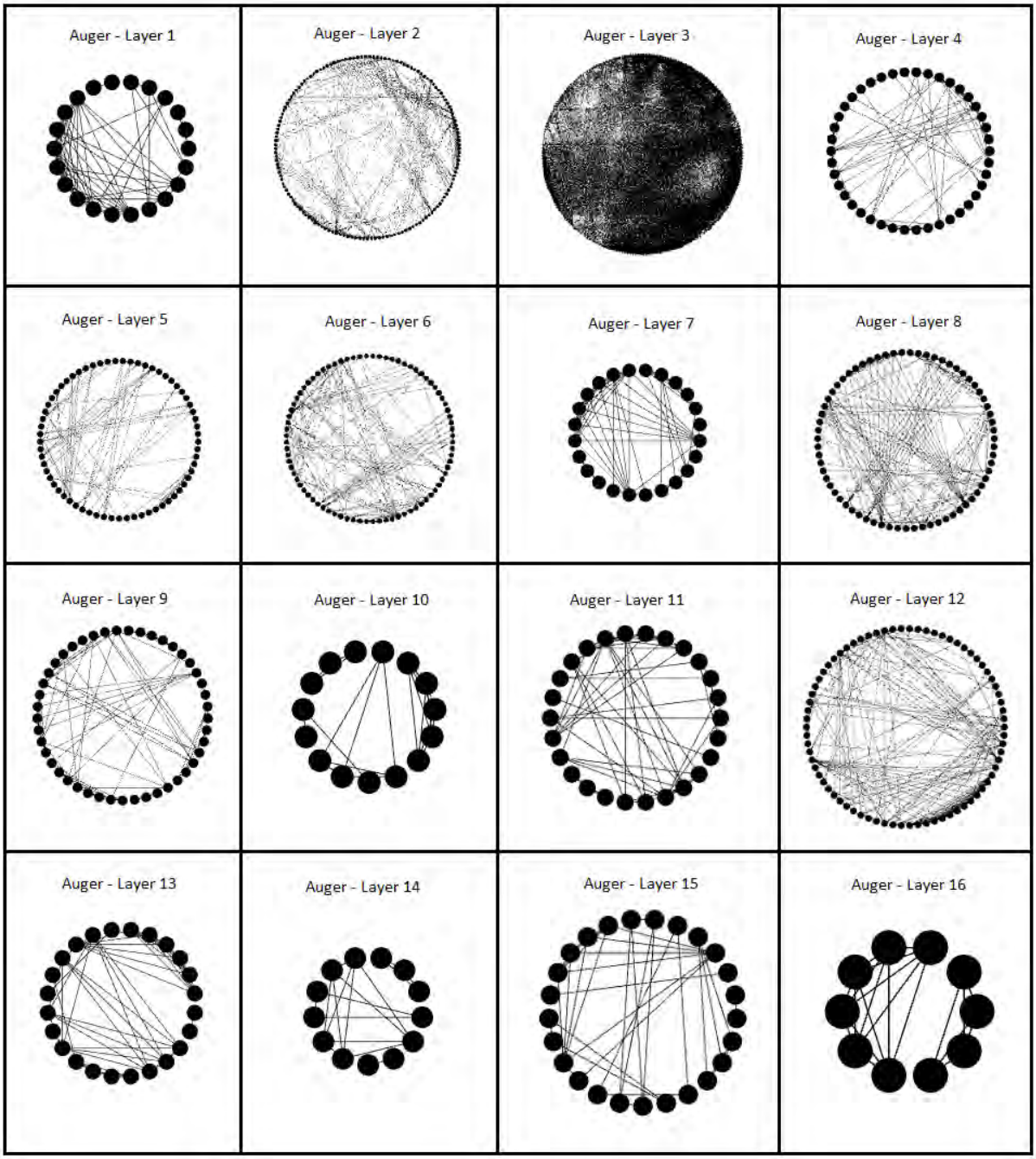
Αριθμός ακμών: 14.180

Αριθμός κορυφών: 475

Πλήθος layer: 16

Παρακάτω βρίσκεται η απεικόνιση του δικτύου:

Εικόνα 50 - Απεικόνιση του δικτύου Auger



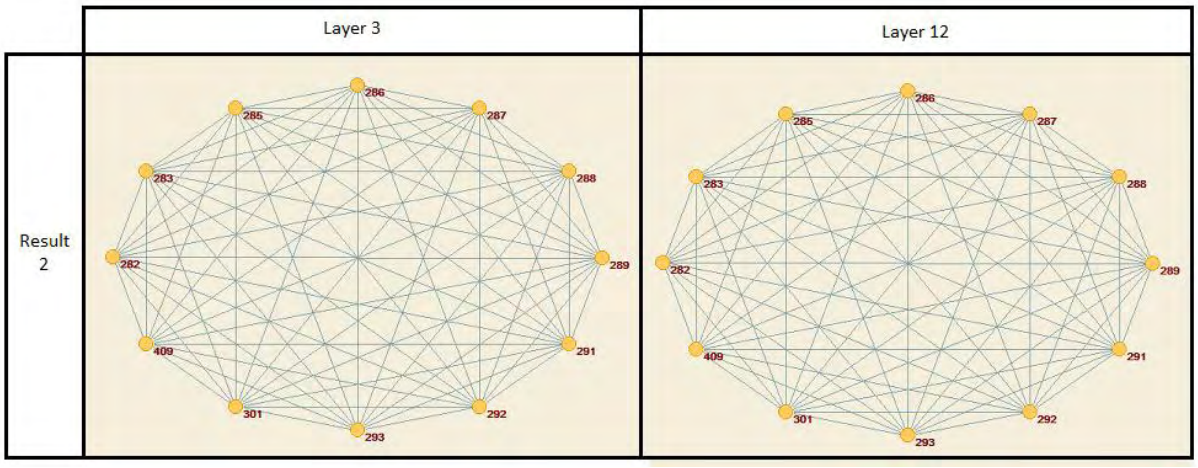
Πηγή: Ιδία επεξεργασία

Τα αποτελέσματα του δικτύου είναι οι κοινότητες:

1) Cluster with O: (125, 1, 85, 239, 257, 84, 81, 99, 82) Layers: (2, 3) Quality: 13.5
2) Cluster with O: (291, 287, 282, 286, 289, 292, 285, 288, 293, 409, 301, 283)

Layers: (12, 3) Quality: 48.0

Εικόνα 51 - Παραδειγμα αποτελέσματος 2



Πηγή: Ιδία επεξεργασία

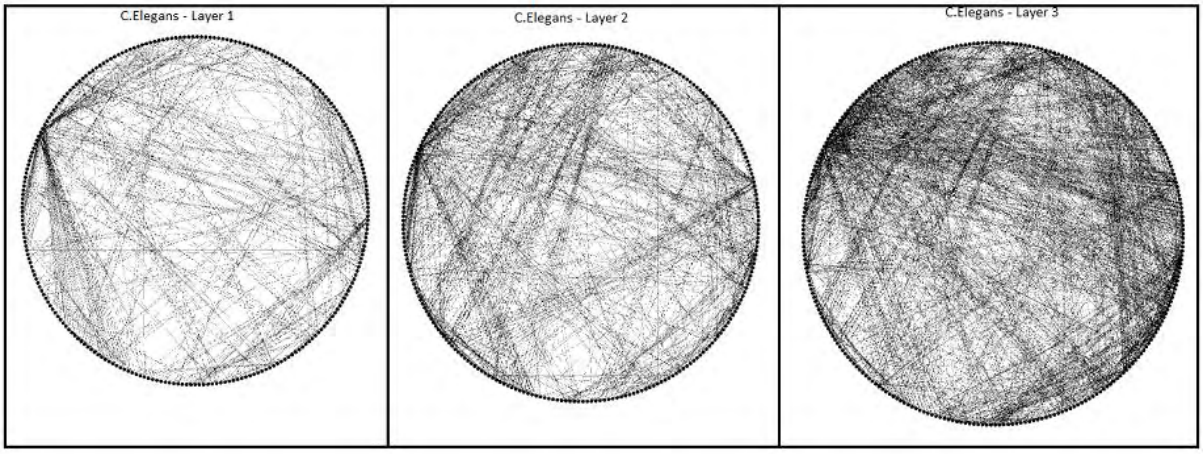
8.4 3ο Δίκτυο

“C.Elegans_largest.txt”[13][14]**:**

Ομοίως με το 1ο δίκτυο (Arabidopsis_largest), μελετιούνται διαφορετικοί τύποι γενετικών αλληλεπιδράσεων για οργανισμούς, με όμοια layer.

Αριθμός ακμών = 5863
Αριθμός κορυφών: 279
Πλήθος layer: 3

Εικόνα 52 - Απεικόνιση του δικτύου C.Elegans

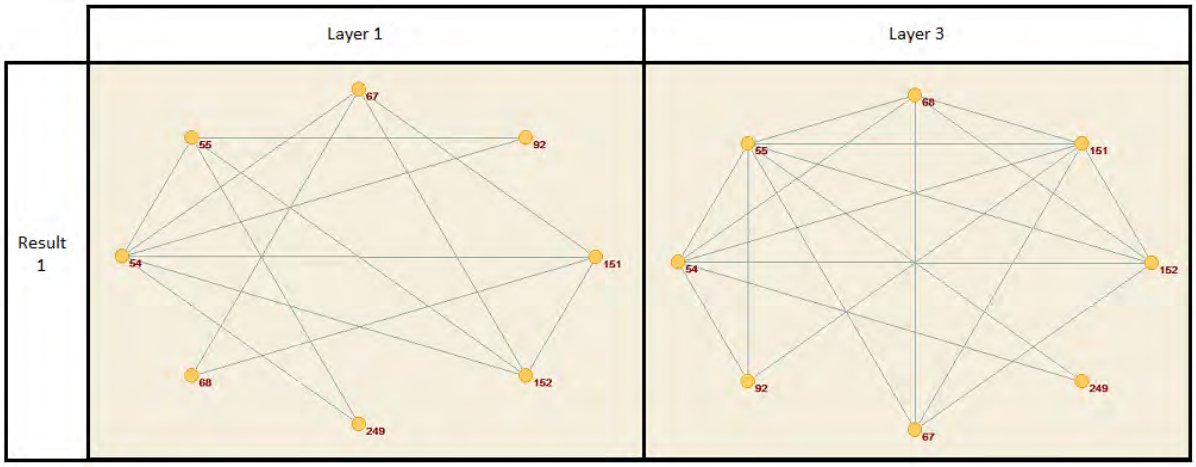


Πηγή: Ιδία επεξεργασί

Τα αποτελέσματα του δικτύου είναι οι κοινότητες:

1) Cluster with O: (54, 55, 152, 151, 67, 68, 249, 92) Layers: (1, 3) Quality: 9.1428
2) Cluster with O: (54, 55, 152, 151, 67, 68, 92, 38) Layers: (1, 3) Quality: 9.1428
3) Cluster with O: (54, 55, 152, 151, 67, 249, 92, 38) Layers: (1, 3) Quality: 9.1428
4) Cluster with O: (54, 55, 152, 151, 67, 68, 249, 38) Layers: (1, 3) Quality: 9.1428
5) Cluster with O: (54, 55, 152, 151, 67, 68, 38, 248) Layers: (1, 3) Quality: 9.1428
6) Cluster with O: (54, 55, 152, 151, 67, 92, 38, 248) Layers: (1, 3) Quality: 9.1428
7) Cluster with O: (54, 55, 152, 151, 249, 92, 38, 248) Layers: (1, 3) Quality: 9.1428
8) Cluster with O: (54, 55, 152, 151, 67, 249, 38, 248) Layers: (1, 3) Quality: 9.1428
9) Cluster with O: (54, 55, 152, 151, 67, 68, 92, 248) Layers: (1, 3) Quality: 9.1428
10) Cluster with O: (54, 55, 152, 151, 67, 249, 92, 248) Layers: (1, 3) Quality: 9.1428
11) Cluster with O: (54, 55, 152, 151, 67, 68, 249, 248) Layers: (1, 3) Quality: 9.1428

Εικόνα 53 - Παραδειγμα αποτελέσματος 3



Πηγή: Ιδία επεξεργασία

8.5 4ο Δίκτυο

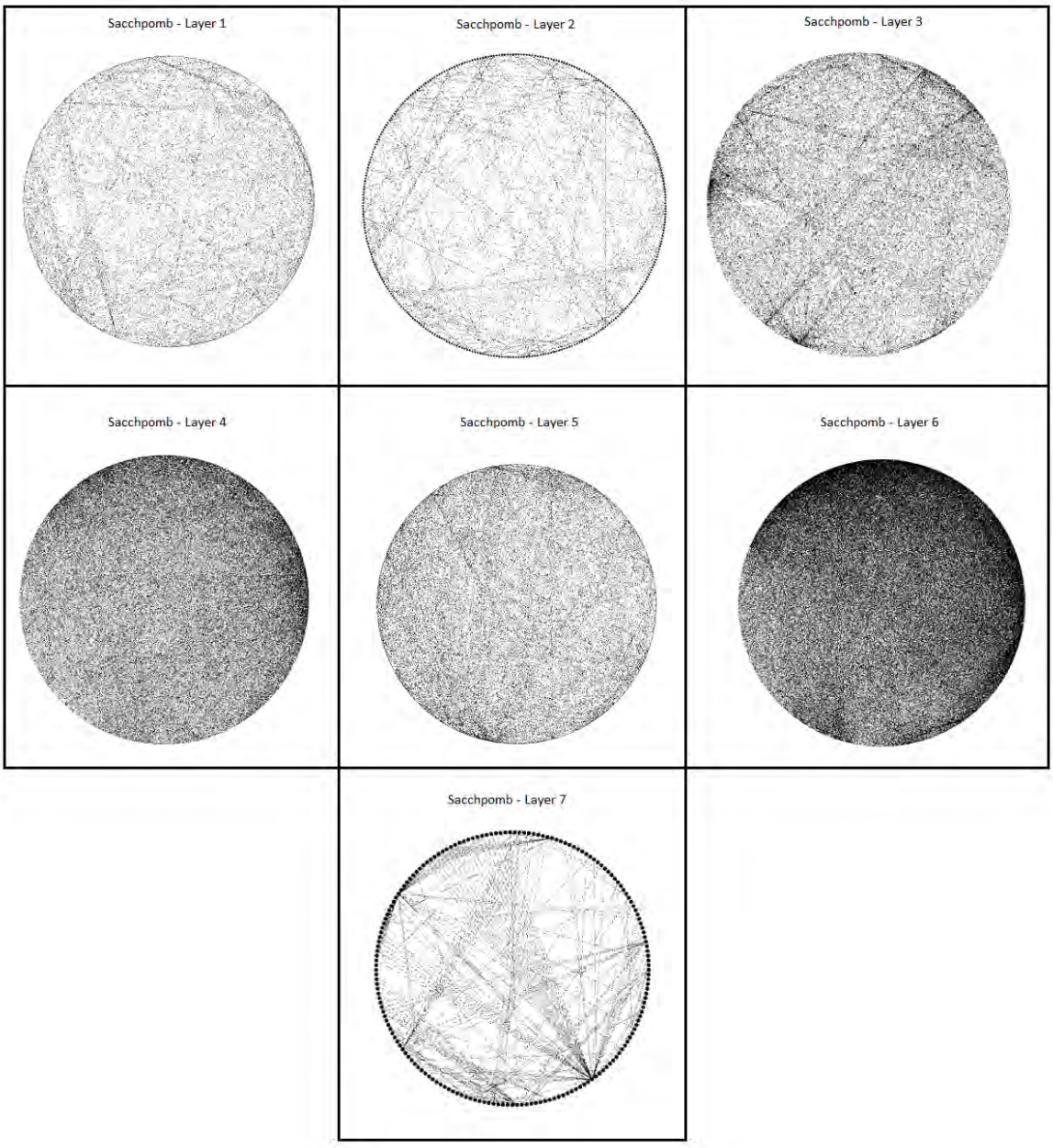
“Sacchpomb_largest.txt”[13][14]**:**

Ομοίως με το 1ο δίκτυο (Arabidopsis_largest), μελετιούνται διαφορετικοί τύποι γενετικών αλληλεπιδράσεων για οργανισμούς, με όμοια layer:

Αριθμός ακμών: 63.401
Αριθμός κορυφών: 4078
Πλήθος layer: 7

Παρακάτω βρίσκεται η απεικόνιση του δικτύου:

Εικόνα 54 - Απεικόνιση του δικτύου Sacchpomb

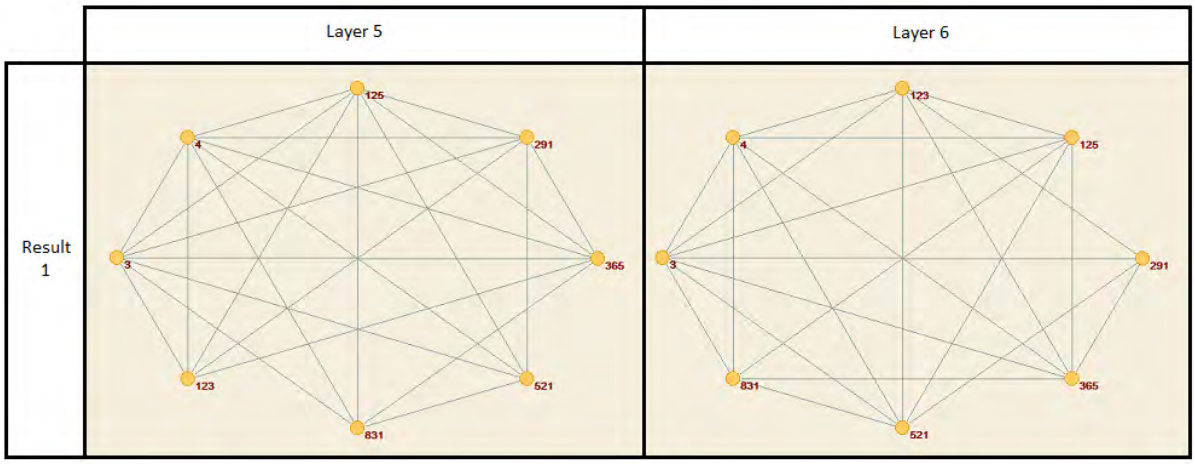


Πηγή: Ιδία επεξεργασία

Τα αποτελέσματα του δικτύου είναι οι κοινότητες:

1) Cluster with O: (365, 125, 521, 4, 3, 291, 831, 123) Layers: (5, 6) Quality: 13.7142
2) Cluster with O: (365, 125, 4, 60, 219, 291, 6, 5, 62, 314) Layers: (5, 6) Quality: 11.1111
3) Cluster with O: (365, 125, 4, 60, 219, 291, 6, 5, 314, 56) Layers: (5, 6) Quality: 11.1111
4) Cluster with O: (365, 125, 4, 60, 219, 291, 6, 5, 62, 56) Layers: (5, 6) Quality: 11.1111
5) Cluster with O: (327, 671, 84, 475, 658, 657, 287, 1413) Layers: (3, 4) Quality: 10.2857

Εικόνα 55 - Παράδειγμα αποτελέσματος 4



Πηγή: Ιδία επεξεργασία

8.6 Περαιτέρω δίκτυα και αποτελέσματα

Έγινε αναζήτηση κοινοτήτων και σε άλλα δίκτυα, τα οποία δεν παρουσιάζονται λόγω έλλειψης αποτελεσμάτων με ενδιαφέρον. Στην κατηγορία αυτή βρίσκονται τα δίκτυα που δεν περιέχουν κανένα ενδιαφέρον cluster, όπως επίσης και εκείνα που είναι πολύ μεγάλα ώστε να μπορέσει να βρεθεί κάποια, προσεγγιστική τουλάχιστον, λύση με τη χρήση του λογισμικού.

Πίνακας 1 – Συνοπτικά αποτελέσματα πειραμάτων

	Κορυφές	Ακμές	Layers	Maximal cluster που βρέθηκαν	Χρόνος
Arabidopsis	6,692	17,930	7	12	29 λεπτά 33 δευτ.
Auger*	475	14,180	16	2	~ 5.5 ώρες
C.Elegans*	279	5,863	3	11	~ 3.5 ώρες
Sacchpomb*	4,078	63,401	7	5	~ 9 ώρες
EUAir	417	3,588	37	0	44 δευτ.
London	369	441	3	0	7 δευτ.
Mus	7,402	19,326	7	0	1 λεπτό 44 δευτ.
Sacchere	6,567	280,849	7	x	> 48 ώρες
Homo	18,136	168,525	7	x	> 48 ώρες
NYClimate	99,666	348,585	3	x	> 48 ώρες
Arxiv	8,798	97,314	13	x	> 48 ώρες
MoscowAthletics2013	81,334	202,686	3	x	> 48 ώρες

Πηγή: Ιδία επεξεργασία

Στα multi-layer δίκτυα του πίνακα που έχουν το σύμβολο (*) έχει βρεθεί μια προσεγγιστική λύση από το λογισμικό λόγω του μεγάλου μεγέθους των δικτύων. Το λογισμικό είναι εξαιρετικά γρήγορο στην ανάλυση των δικτύων που δεν περιέχουν ενδιαφέροντα cluster γιατί γίνεται κλάδεμα στα αρχικά υποδέντρα και το εύρος αναζήτησης μειώνεται σημαντικά.

9. Σύνοψη – Αδυναμίες λογισμικού / Μελλοντικές επεκτάσεις

9.1 Σύνοψη

Η χρήση των quasi-cliques στα σύνθετα δίκτυα βοηθάει στο να βρεθούν κοινότητες των οντοτήτων που υπάρχουν μέσα σε αυτό, που συνήθως αναπαριστούν στοιχεία με παρόμοια χαρακτηριστικά. Στα σύγχρονα σύνθετα δίκτυα είναι μια αρκετά δαπανηρή διαδικασία όσον αφορά το κόστος υπολογισμού της και πολλές φορές είναι δυνατό να προκύπτει μεγάλο μέρος πλεονάζουσας πληροφορίας. Για το λόγο αυτό, έγινε χρήση των μεθόδων που αναπτύχθηκαν παραπάνω, όπως επίσης και του κλαδέματος που βελτιώνει σημαντικά την απόδοση της συνολικής διαδικασίας. Ακόμα, χρησιμοποιήθηκε η πλατφόρμα για cluster computing Spark, που εξελίσσεται ραγδαία και καθιστά ικανή την αποτελεσματική διαχείριση και επεξεργασία για big data. Το λογισμικό που δημιουργήθηκε στα πλαίσια της παρούσας εργασίας είναι ικανό να υπολογίσει τις κοινότητες σε ένα σύνθετο δίκτυο, σύμφωνα με τις προδιαγραφές (μέγεθος, πυκνότητα, ανεκτή επικάλυψη) που ορίζουμε για κάθε δίκτυο.

9.2 Αδυναμίες λογισμικού και μελλοντικές επεκτάσεις

Παρόλο που το λογισμικό μπορεί να ανακαλύψει τις κοινότητες που υπάρχουν σε ένα σύνθετο δίκτυο, υπάρχουν κάποιες αδυναμίες που πρέπει να αναφερθούν. Το λογισμικό δεν μπορεί να υποστηρίξει πολύ μεγάλου μεγέθους δίκτυα και μπορεί να τελειώσει η μνήμη του πριν την αίσια ολοκλήρωση του. Σε κάποιες περιπτώσεις "προλαβαίνει" να βρει μερικές κοινότητες πριν τερματίσει λόγω έλλειψης μνήμης, ενώ σε κάποιες άλλες όχι. Δεν είναι δυνατό να προσδιορίσουμε ποια ακριβώς θεωρούνται πολύ μεγάλα δίκτυα για το λογισμικό για τον λόγο ότι αυτό εξαρτάται και από άλλους παράγοντες πέρα από τις ακμές και τις κορυφές. Πολύ μεγάλο ρόλο παίζει η συνδεσμολογία. Για παράδειγμα, μπορεί ένα δίκτυο να περιέχει αρκετές χιλιάδες ακμές σε ένα μόνο layer και μόνο λίγες στα άλλα. Αυτές οι χιλιάδες ακμές είναι πολύ πιθανό να κλαδευτούν σε πολύ αρχικό στάδιο και να απομείνει ένα μικρό μέρος του δικτύου, αυτό που βρίσκεται σε παραπάνω από ένα layer, για να αναλυθεί.

Στο λογισμικό έχουν υλοποιηθεί κάποιες τεχνικές κλαδέματος, που όμως δεν είναι αρκετές καθώς στα μεγάλα δίκτυα έχουμε μεγάλους χρόνους εκτέλεσης. Επίσης, αν υλοποιηθούν περισσότερες τεχνικές κλαδέματος, θα βελτιωθούν, εκτός από τους χρόνους εκτέλεσης και οι περιπτώσεις όπου έχουμε έλλειψη μνήμης.

Περισσότερες και πιο σύνθετες μέθοδοι κλαδέματος μπορούν να υλοποιηθούν σε μελλοντική έκδοση του λογισμικού. Μελλοντικές επεκτάσεις μπορούν να χρησιμοποιούν παραπάνω αλγορίθμους ανακάλυψης κοινοτήτων ανάλογα με το ποιος είναι ταχύτερος για το εκάστοτε δίκτυο.

Σε μελλοντική έκδοση μπορεί, επιπλέον, να προστεθεί γραφικό περιβάλλον, όπου κανείς θα μπορεί να παρακολουθεί την εκτέλεση και να μπορεί να δει τα αποτελέσματα σαν γραφήματα, καθώς ακόμα και να μπορεί να επεξεργαστεί τις παραμέτρους εκτέλεσης (επιθυμητό μέγεθος cluster, αποδεκτής πυκνότητας και επικάλυψης).

10. Παράρτημα

Στο παράτημα παρατίθεται ο κώδικας του λογισμικού MCD.

```
package community_detection_pkg
import org.apache.log4j.{Level, Logger}
import org.apache.spark._
import org.apache.spark.graphx.{EdgeTriplet, _}
import org.apache.spark.rdd.RDD
//import for mutable List Buffer
import scala.collection.mutable.ListBuffer
//import for queue and array and map
import scala.collection.mutable.Queue
//import breaks
import scala.util.control.Breaks._
/**
  * Created by Lazaros Koiou on 28/9/2016.
  */
class node(graph:Graph[Any, String], num_layer:Int)
{
  var O = new ListBuffer[(graphx.VertexId)]  // set of vertices
  var Q :Double = 0
  var Qc:Double = 0
  var isCluster :Boolean = false
  var So = new ListBuffer [String] ()        // cand layers
  var S = new ListBuffer [String] ()
  var valid:Boolean = false
  var cand:Array[ListBuffer[(graphx.VertexId)]] = new Array[ListBuffer[(graphx.VertexId)]](num_layer)
  def root_cand(): Unit =
  {
    println("\nRoot candidates:")
    for (layer <- So.sortWith(_ < _)) {
      val pos:Int = layer.toInt
      var list:ListBuffer[VertexId] = new ListBuffer[VertexId]
      graph.edges.collect
        .foreach {Edge =>
          if (!list.contains(Edge.srcId) && (Edge.attr == layer.toString))
          {
            list += Edge.srcId
            if (!So.contains(layer.toString)) So += layer.toString
          }
          else if (!list.contains(Edge.dstId) && (Edge.attr == layer.toString))
          {
            list += Edge.dstId
            if (!So.contains(layer.toString)) So += layer.toString
          }
        }
      cand(pos-1) = list.sortWith(_ < _)
      println("candO," + pos + " = " + cand(pos - 1).sortWith(_ < _))
    }
    So = So.sortWith(_ < _)
    println("\nSo from root_cand:" + So)
  }
```

```
  def make_cand(u:graphx.VertexId): Unit =
  {
    //u vertex is used to form the new candidates list
    for (layer <- So.sortWith(_ < _)) {
      var list: ListBuffer[VertexId] = new ListBuffer[VertexId]
      /* Must:
      * 1. Not be contained in list (no extra copies)
      * 2. Contained in relevant layer
      * 3. Be a neighboring vertex to u that we have added in STnext
      * 4. Don't add to cand ea vertex that exists in node (O)
      */
      if (O.length == 1)
      {
        graph.edges.collect
          .foreach { Edge =>
            if (!list.contains(Edge.srcId) && (Edge.attr == layer.toString) && O.contains(Edge.dstId)
&& !O.contains(Edge.srcId)) {
              list += Edge.srcId
              if (!So.contains(layer.toString)) So += layer.toString
            }
            else if (!list.contains(Edge.dstId) && (Edge.attr == layer.toString) && O.contains(Edge.srcId)
&& !O.contains(Edge.dstId)) {
              list += Edge.dstId
              if (!So.contains(layer.toString)) So += layer.toString
            }
          }
        cand(layer.toInt - 1) = list
      }
      else
      {
        list = cand(layer.toInt - 1)
        graph.edges.collect
          .foreach { Edge =>
            if (!list.contains(Edge.srcId) && (Edge.attr == layer.toString) && (u == Edge.dstId)
&& !O.contains(Edge.srcId)) {
              list += Edge.srcId
            }
            else if (!list.contains(Edge.dstId) && (Edge.attr == layer.toString) && (u == Edge.srcId)
&& !O.contains(Edge.dstId)) {
              list += Edge.dstId
            }
          }
        cand(layer.toInt - 1) = list
      }
    }
  }
  def find_max_cand(): VertexId =
  {
    var i:Int = 0
    //create a mutable Map that has (VertexId), (Σ deg(u))
    var cand_map = scala.collection.mutable.Map[VertexId, Int]()
    // for each layer make a mini subgraph of the subgraph to find max candidate
    for (layer <- So.sortWith(_ < _)) {
      val layergraph: Graph[Any, String] = graph.subgraph(epred = e => layer == e.attr)
      val degrees: VertexRDD[Int] = layergraph.degrees
      if (O.isEmpty)  //if root
      {
        degrees.collect.foreach {
          case (vertex, deg) =>
            if (!cand_map.keySet.contains(vertex)&& cand(layer.toInt - 1).contains(vertex)) {
              cand_map += (vertex -> deg)
            }
            else if (cand_map.keySet.contains(vertex)) {
              cand_map.keys.foreach(i =>
                if (i == vertex) {
```

```
                        cand_map(i) += deg
                    }
                )
            }
        }
    }
    else            //if not root
    {
        degrees.collect.foreach {
            case (vertex, deg) =>
                if (!cand_map.keySet.contains(vertex) && cand(layer.toInt - 1).contains(vertex)) {
                    cand_map += (vertex -> deg)
                }
                else if (cand_map.keySet.contains(vertex)) {
                    cand_map.keys.foreach(i =>
                        if (i == vertex) {
                            cand_map(i) += deg
                        }
                    )
                }
        }
    }
    i += 1
}
val maxKey = cand_map.maxBy(_._2)
val u:VertexId = maxKey._1
println("Max Cand Vertex is: " + u + " " + maxKey)
u
}
}

object MCD {
    //thresholds for |O|<8 and |S|<2
    val O_threshold:Int = 8
    val S_threshold:Int = 2
    //we need queue in scope for expand_procedure
    var queue = Queue[node]()
    var result:ListBuffer[node] = new ListBuffer[node]
    var max_layerROOT:Int = -1
    def main(args: Array[String]): Unit = {
        val t0 = System.nanoTime()
        val conf = new SparkConf()
            .setAppName("Load graph")
            .setMaster("local")
        val sc = new SparkContext(conf)
        // stop INFO spark messages
        val rootLogger = Logger.getRootLogger()
        rootLogger.setLevel(Level.ERROR)
        val inputFile = "/home/hm/Desktop/testnet_inputs/tested/Arabidopsis_largest.txt"
        // data form must be:
        // layer vertex_source vertex_dest
        val edges: RDD[Edge[String]] =
            sc.textFile(inputFile).map { line =>
                val fields = line.split(" ")
                Edge(fields(1).toLong, fields(2).toLong, fields(0))
            }.distinct
        val graph: Graph[Any, String] = Graph.fromEdges(edges, "V")
        graph.cache()
        println("num edges = " + graph.numEdges)
        println("num vertices = " + graph.numVertices)
        //create root ST
        println("-----")
        println("Creating root ST...")
        var Sroot = new ListBuffer[String]()
        for (triplet <- graph.triplets.collect) {
            if (!Sroot.contains(triplet.attr: String)) {
                Sroot += (triplet.attr: String)
```

```
    }
}
Sroot = Sroot.sortWith(_ < _)
for (temp <- Sroot) {
    if (max_layerROOT < temp.toInt) {
        max_layerROOT = temp.toInt
    }
}
println("ROOT max layer is: " + max_layerROOT)
val root = new node(graph, max_layerROOT)
root.So = Sroot
root.root_cand() //only for ROOT
println("-----")
var continue = true
var vertices_num_bef = 0
var vertices_num_after = 0
for (layer <- root.So)
{
    for (c <- root.cand(layer.toInt - 1))
    {
        vertices_num_bef += 1
    }
}
println("Pruning....")
while (continue){
    continue = false
    for (layer <- root.So) {
        val OcandO: ListBuffer[VertexId] = new ListBuffer[VertexId]
        //prune all candidates that belong to less than 2 layers
        prune1(root, layer)
        for (c <- root.cand(layer.toInt - 1)) {OcandO += c}
        if (root.cand(layer.toInt - 1).nonEmpty) {
            //prune all candidates that have degree less than ceil(γ * (|O| - 1))
            if (prune_ceil(root, layer, OcandO, graph))
                {
                    continue = true
                }
        }
        OcandO.clear()
    }
}
for (layer <- root.So)
{
    for (c <- root.cand(layer.toInt - 1))
    {
        vertices_num_after += 1
    }
}
val pruned_vertices = vertices_num_bef - vertices_num_after
println("Number of vertices pruned from candO,i: " + pruned_vertices)
println("\nS.cand after pruning:")
for (layer <- Sroot)
    {
        println("cand("+layer+"): " + root.cand(layer.toInt - 1))
    }
//initialize queue with ROOT
println("Initializing Queue...")
println("-----")
queue += root
var step:Int = -1
//while end condition is empty queue
while (queue.nonEmpty) {
    step += 1
        //queue sorted before pop
        queue = queue.sortWith(_.Q > _.Q)
        val Obj = queue.dequeue()
        println("\nStep " + step + ": Current node is: " + Obj.O.toList + " with Q = " + Obj.Q)
```

```
        println("Sum of queue elements with quality " + Obj.Q + " is " + queue.count(_.Q == Obj.Q))
    if (!Obj.isCluster) {
        println("Current node is subtree!")
        val u: VertexId = Obj.find_max_cand()
        //expand subtree
        expand(graph, u, Obj)
        println("Queue has " + queue.length + " elements")
        println("Result has " + result.length + " elements")
        if ((step%20 == 0) && result.nonEmpty)
        {
            for (i<-result)
            {
                println("Cluster with O: " + i.O.sortWith(_ < _) + " S: "+i.S + " Q: " + i.Q)
            }
        }
        println("\n-----")
    }
//
    else
    {
    println("Current node is cluster!")
    //check if redundant if false then add it to Result List
    var redundant: Boolean = false
    var temp: Boolean = true
          if (result.nonEmpty) {
              breakable {
                  for (i <- 0 until result.length) {
                      temp = checkifRedundant(Obj, result(i), graph)
                      if (temp) {
                          redundant = true
                      }
                      if (temp) break()
                  }
              }
              if (!redundant) {
                  //if not redundant add it to result
                  if ((Obj.S.length >= S_threshold) && (Obj.O.length >= O_threshold) ) {
                      result += Obj
                      println("***Cluster added to Result***")
                      println("Checking if there are redundant clusters in result...")
                      //find redundant clusters in result
                      var temp:ListBuffer[node] = new ListBuffer[node]
                      for (r <- result.sortWith(_.Q > _.Q))
                      {
                          var isRedundant = false
                          result -= r
                          if (temp.nonEmpty)
                          {
                              breakable  { for (t <- temp)
                              {
                                  if (checkifRedundant(r, t, graph))
                                  {
                                      isRedundant = true
                                      println("Redundant cluster found in result...Keeping the one with better quality!")
                                      break()
                                  }
                              } }
                          }
                          if (!isRedundant)
                          {
                              temp += r
                          }
                      }
                      for (t <- temp)
                      {
                          temp -= t
                          result += t
```

```
                        }
                        //end find redundant clusters in result
                        println("New result contains:")
                        for (i<-result)
                            {
                                println("Cluster with O: " + i.O + " S: "+i.S+ " Q: " + i.Q)
                            }
                    }
                }
            }
            else {
                // if result is empty then cluster cant be redundant
                if ((Obj.S.length >= S_threshold) && (Obj.O.length >= O_threshold) ) {
                    result += Obj
                    println("***Cluster added to Result***")
                    println("New result contains:")
                    for (i<-result)
                    {
                        println("Cluster with O: " + i.O + " S: "+i.S+ " Q: " + i.Q)
                    }
                }
            }
            println("-----")
            println("Queue has " + queue.length + " elements")
            println("Result has " + result.length + " elements")
            if ((step%20 == 0) && result.nonEmpty)
                {
                    for (i<-result)
                    {
                        println("Cluster with O: " + i.O + " S: "+i.S + " Q: " + i.Q)
                    }
                }
            println("\n-----")
        }
    }//end of while loop
    var count: Int = 1
    println("\nRESULT = ")
    for (r <- result) {
        println(count + ") Cluster: " + r.O.sortWith(_ < _) + "\t S: " + r.S.sortWith(_ < _) + "\t Q: " + r.Q)
        count += 1
    }
    //calculate time
    val t1 = System.nanoTime()
    val time_in_sec = ((t1 - t0) / 1000000000).toInt
    val hours = (time_in_sec / 3600).floor
    val minutes = ((time_in_sec - (hours * 3600))/60).floor
    val seconds = time_in_sec - (hours * 3600) - (minutes * 60)
    println("\nElapsed time: " + time_in_sec.toInt + " seconds")
    println("Elapsed time: " + hours.toInt + " hours " + minutes.toInt + " minutes " + seconds.toInt + " seconds")
}
def checkifRedundant(C1:node, Cresult:node, graph:Graph[Any, String] ): Boolean =
{
    var cond1:Boolean = false
    var cond2:Boolean = false
    val r:Double = 0.25
    //if the cluster already exists in result
    if ((C1.O.sortWith(_ < _) == Cresult.O.sortWith(_ < _)) && (C1.S.length <= Cresult.S.length) && (C1.Q <=
Cresult.Q))
    {
        println("***Cluster is redundant because it already exists in result!***")
        return true
    }
    //condition 1
    if (C1.O.sortWith(_ < _) != Cresult.O.sortWith(_ < _))
        {
            cond1 = true
        }
```

```
        if (!cond1) return false
        //condition 2
        if (C1.Q < Cresult.Q)
            {
                cond2 = true
            }
        if (!cond2) return false
        //condition 3
        //section that of S and S'
        var i: ListBuffer[String] = new ListBuffer[String]()
        for(j <- C1.S ; k <- Cresult.S)
            {
                if (j == k)
                    {
                        i += j
                    }
            }
        i = i.sortWith(_ < _)
        val C1_Olist = C1.O.toList
        val C2_Olist = Cresult.O.toList
        // find edge overlaps between 2 clusters
        var iEi:Double = 0
        var sum:Double = 0
        for(layer <- i)
          {
            val subgraph1:Graph[Any, String] = graph.subgraph(vpred = (v, d) => C1_Olist.contains(v), epred = e => layer ==
e.attr)
            val subgraph2:Graph[Any, String] = graph.subgraph(vpred = (v, d) => C2_Olist.contains(v), epred = e => layer ==
e.attr)
            iEi = 0
            for (triplet1 <- subgraph1.triplets.collect ; triplet2 <- subgraph2.triplets.collect)
                {
                    if (triplet1.equals(triplet2))
                        {
                            iEi +=1
                        }
                }
            sum += iEi / subgraph1.numEdges
          }
        val EdgeOverlap:Double = sum * 1 / C1.S.length
        if ((EdgeOverlap >= r) && cond1 && cond2)
            {
                println("1/S * (iEi / Ei(O)) = " + EdgeOverlap + " >= r = 0.25")
                println("Cluster is redundant!\n")
                true
            }
        else false
    }
    def expand(graph:Graph[Any, String], u:graphx.VertexId, STremain:node): Unit = {
        var Onext = new ListBuffer[(graphx.VertexId)]
        var Sonext = new ListBuffer[String]()
        var candnext: Array[ListBuffer[(graphx.VertexId)]] = new Array[ListBuffer[(graphx.VertexId)]](max_layerROOT)
        var recycle = false
        val validation_num = 8
        println("Expanding " + STremain.O + " subtree...")
        for (elem <- STremain.O) {
            Onext += elem
        }
        Onext += u
        for (elem <- STremain.So) {
            Sonext += elem
        }
        //pass cand to cand_next
        for (layer <- Sonext) {
            var listnext: ListBuffer[VertexId] = new ListBuffer[VertexId]
            for (elem <- STremain.cand(layer.toInt - 1)) {
                listnext += elem
```

```
            }
            candnext(layer.toInt - 1) = listnext
        }
        //calc Sonext
        for (layer <- Sonext) {
            if (!candnext(layer.toInt - 1).contains(u)) {
                Sonext -= layer
            }
        }
        var same_next = false
if ((Onext.length <= validation_num)||STremain.valid)
    {//this is made to minimize search area, search |O|>8 ONLY when a cluster is formed for 8 vertices
        breakable {
            for (q <- queue) {
                if (Onext.sortWith(_ < _) == q.O.sortWith(_ < _) && (Sonext.length <= q.So.length)) //if same cluster exists dont
insert to queue
                {
                    same_next = true
                    break()
                }
            }
        }
        if (!same_next) {
            //make a new cluster with Onext and Sonext
            var STnext: node = new node(graph, max_layerROOT)
            STnext.O = Onext
            STnext.So = Sonext.sortWith(_ < _)
            for (layer <- STnext.So) {
                if (candnext(layer.toInt - 1).contains(u)) {
                    candnext(layer.toInt - 1) -= u
                }
                STnext.cand(layer.toInt - 1) = candnext(layer.toInt - 1)
            }
            //STnext.cand must be made again from scratch for vertex u
            STnext.make_cand(u)
            println("\n->STnext cluster/subtree created with .O = " + STnext.O + " .So = " + STnext.So)
            println("before prune:")
            for (layer <- STnext.So.sortWith(_ < _)) {
                println("candO," + layer + ": " + STnext.cand(layer.toInt - 1))
            }
            var v_next_before = 0
            var v_next_after = 0
            for (layer <- STnext.So) {
                for (c <- STnext.cand(layer.toInt - 1)) {
                    v_next_before += 1
                }
            }
            var toprune: Boolean = false
            var continue: Boolean = true
            while (continue) {
                continue = false
                for (layer <- STnext.So) {
                    val OcandO: ListBuffer[VertexId] = new ListBuffer[VertexId]
                    //prune all candidates that belong to less than 2 layers
                    prune1(STnext, layer)
                    for (c <- STnext.cand(layer.toInt - 1)) {
                        OcandO += c
                    }
                    for (c <- STnext.O) {
                        OcandO += c
                    }
                    if (STnext.cand(layer.toInt - 1).nonEmpty) {
                        //prune all candidates that have degree less than ceil(γ * (|O| - 1))
                        if (prune_ceil(STnext, layer, OcandO, graph)) {
                            continue = true
                        }
                        //prune layers in which O contains a vertex with degree less than ceil(γ * (|O| - 1))
```

```
            toprune = prune2(STnext.O, layer, OcandO, graph)
            if (toprune) {
                STnext.So -= layer
                continue = true
            }
        }
        OcandO.clear()
    }
}
println("after prune:")
for (layer <- STnext.So.sortWith(_ < _)) {
    println("candO," + layer + ": " + STnext.cand(layer.toInt - 1))
}
for (layer <- STnext.So) {
    for (c <- STnext.cand(layer.toInt - 1)) {
        v_next_after += 1
    }
}
val v_next_pruned = v_next_before - v_next_after
println("pruned from cand: " + v_next_pruned)
var end_subtree_next = true
breakable {
    for (layer <- STnext.So) {
        if (STnext.cand(layer.toInt - 1).nonEmpty) {
            end_subtree_next = false
            break()
        }
    }
}
if (!end_subtree_next) //if there are no more candidates dont add to queue
{
    println("Calculating Qest...")
    calc_Qest(STnext, graph)
    println("STnext Qest():" + STnext.Q)
    if (STnext.Q >= 0) {
            queue += STnext
            println("***Subtree added to queue***")
    }
    else println("***Q < 0. Subtree not added to queue***")
    recycle = true
}
else {
    println("***Subtree end. No more candidates***")
    recycle = true
    println("***Recycled***")
}
//Created ONLY if quality is > 0
//Checking if there is a cluster C = (Onext, S) and if redundant
println("\nChecking if cluster C = (Onext, S) exists... , where S c So")
println("Onext = " + STnext.O + " S = " + STnext.So)
val s_graph: Graph[Any, String] = graph.subgraph(vpred = (v, d) => Onext.contains(v))
calc_quality(STnext, s_graph)
if (STnext.Qc > 0) {
    var c: node = new node(graph, max_layerROOT)
    for (o <- STnext.O) {
        c.O += o
    }
    for (s <- STnext.S) {
        c.S += s
    }
    c.Q = STnext.Qc
    c.isCluster = true
    //this is to initialize cand as empty list instead of null
    var list: ListBuffer[VertexId] = new ListBuffer[VertexId]
    list += 1
    list -= 1
    for (layer <- c.S) {
```

```
                c.cand(layer.toInt - 1) = list
            }
            //if not redundant add cluster to queue
            var redundant: Boolean = false
            var temp: Boolean = true
            if ((c.Q > 0) && (c.S.length >= S_threshold) && (c.O.length >= O_threshold)) {
                if (result.nonEmpty) {
                    breakable {
                        for (i <- 0 until result.length) {
                            temp = checkifRedundant(c, result(i), graph)
                            if (temp) {
                                redundant = true
                                println("***Cluster is redundant!***")
                                c = null
                                break()
                            }
                        }
                    }
                    if (!redundant) {
                        queue += c
                        println("***Cluster added in queue!***")
                        STnext.valid = true
                    }
                }
                else {
                    queue += c
                    println("***Cluster added in queue!***")
                    STnext.valid = true
                }
            }
            else {
                c = null
                println("***Not a cluster!***")
            }
        }
        else {
            println("***Cluster not created***")
        }
        if (recycle) {
            STnext = null
        }
    }
    else {
        println("***Subtree " + Onext + " was the same as one which is already in queue***")
        println("\n***Cluster is the same as one which is already in queue***")
    }
}
else
    { println("\n***|O| > " +validation_num+ " in "+ Onext +". Pruned to minimize search tree***") }
    for (layer <- STremain.So) {
        STremain.cand(layer.toInt - 1) -= u
    }
    if ((STremain.O.length <= validation_num) || STremain.valid ) {
        println("\n->STremain cluster/subtree created with .O = " + STremain.O + " .So = " + STremain.So)
        println("before prune:")
        for (layer <- STremain.So.sortWith(_ < _)) {
            println("candO," + layer + ": " + STremain.cand(layer.toInt - 1))
        }
        var v_remain_before = 0
        var v_remain_after = 0
        for (layer <- STremain.So) {
            for (c <- STremain.cand(layer.toInt - 1)) {
                v_remain_before += 1
            }
        }
        var toprune_remain: Boolean = false
        var continue_remain: Boolean = true
```

```
while (continue_remain) {
    continue_remain = false
    for (layer <- STremain.So) {
        val OcandO: ListBuffer[VertexId] = new ListBuffer[VertexId]
        //prune all candidates that belong to less than 2 layers
        prune1(STremain, layer)
        for (c <- STremain.cand(layer.toInt - 1)) {
            OcandO += c
        }
        for (c <- STremain.O) {
            OcandO += c
        }
        if (STremain.cand(layer.toInt - 1).nonEmpty) {
            //prune all candidates that have degree less than ceil(γ * (|O| - 1))
            if (prune_ceil(STremain, layer, OcandO, graph)) {
                continue_remain = true
            }
            //prune all candidates that have degree less than ceil(γ * (|O| - 1))
            toprune_remain = prune2(STremain.O, layer, OcandO, graph)
            if (toprune_remain) {
                STremain.So -= layer
                continue_remain = true
            }
        }
        OcandO.clear()
    }
}
println("after prune:")
for (layer <- STremain.So.sortWith(_ < _)) {
    println("candO," + layer + ": " + STremain.cand(layer.toInt - 1))
}
for (layer <- STremain.So) {
    for (c <- STremain.cand(layer.toInt - 1)) {
        v_remain_after += 1
    }
}
val v_remain_pruned = v_remain_before - v_remain_after
println("pruned from cand: " + v_remain_pruned)
//same subtrees will be avoided
var never_valid = false
var end_subtree_remain = true
breakable {
    for (layer <- STremain.So) {
        if (STremain.cand(layer.toInt - 1).nonEmpty) {
            end_subtree_remain = false
            break()
        }
    }
}
if (STremain.So.length >= S_threshold) //if |So| < 2 don't insert to queue
{
    if ((STremain.So.length == S_threshold) && (STremain.O.length < O_threshold)) {
        breakable {
            for (layer <- STremain.So) //for each layer So has, find if empty cand
            {
                if (STremain.cand(layer.toInt - 1).isEmpty) {
                    never_valid = true //prune it
                    break()
                }
            }
        }
    }
    if (!end_subtree_remain) {
        //if no more candidates left dont add subtree to queue
        if (!never_valid) {
            println("Calculating Qest...")
            calc_Qest(STremain, graph)
```

```
                println("STremain Qest():" + STremain.Q)
                if (STremain.Q >= 0) {
                    queue += STremain
                    println("***Subtree added to queue***")
                }
                else println("***Q < 0. Subtree not added to queue***")
            }
            else {
                println("***Never valid. Prune subtree***")
            }
        }
        else {
            println("***Subtree ended. No more candidates")
        }
    }
    else {
        println("\n***|So| < 2. Subtree was not added to queue***")
    }
}
    else { println ("***|O| > " +validation_num+ " && not_valid " + STremain.O + ". Pruned to minimize search tree***")}
    println("\n-----")
  }
  def calc_quality (C1:node, s_graph:Graph[Any, String]): Unit = {
    var i: Int = 0
    var deleted:ListBuffer[String] = new ListBuffer[String]
    //make an array item for each layer
    val γG: Array[Double] = new Array[Double](max_layerROOT)
    if (s_graph.numVertices < O_threshold || C1.So.length < S_threshold)
    {
      C1.Qc = -1
      return
    }
    //pass So to S
    for (layer <- C1.So) { C1.S += layer}
    // for each layer make a mini subgraph of the subgraph to find minimum degree vertex
    for (layer <- C1.S.sortWith(_ < _)) {
      val layergraph: Graph[Any, String] = s_graph.subgraph(epred = e => layer == e.attr)
      //find min degree
      var min_degree = 999
      var min_vertex: VertexId = 999
      val degrees: VertexRDD[Int] = layergraph.degrees
      degrees.filter { case (vertex, deg) => min_degree >= deg }.collect()
        .foreach { case (vertex, deg) => if (min_degree >= deg) {
          min_degree = deg
          min_vertex = vertex
        }
        }
      i = layer.toInt
      γG(i - 1) = min_degree.toDouble / (layergraph.numVertices.toDouble - 1)
      if (γG(i - 1) < 0.5) {
        C1.S -= i.toString
        deleted += i.toString
      }
    }
    var γS: Double = -1
    if (C1.S.nonEmpty) {
      γS = 0
      for (x <- γG)
      {
        γS += x / C1.S.length
      }
    }
    //if |O| < 3 || |S| < 2
    if (γS != -1)
    {
      if (s_graph.numVertices >= O_threshold && C1.S.length >= S_threshold) {
```

```
            C1.Qc = s_graph.numVertices * C1.S.length * γS
            //        println("s_graph.numVertices = " + s_graph.numVertices + " So.length = " + So.length + " gS = " + γS)
            println("Q = " + C1.Qc)
            println("Deleted from |S|: " + deleted)
          }
          else {
            C1.Qc = -1
            println("Q = " + C1.Qc + ": Vertice quantity or layer quantity not fulfilled!")
            println("|S| = " + C1.S.length +" (Some layers may have been removed if not dense enough)")
            println("|O| = " + s_graph.numVertices)
            println("Deleted from |S|: " + deleted)
          }
        }
        else
        {
          C1.Qc = -1
          println("Q = " + C1.Qc + ": Vertice quantity or layer quantity not fulfilled!")
          println("|S| = " + C1.S.length)
          println("|O| = " + s_graph.numVertices)
          println("Deleted from |S|: " + deleted)
        }
      }
      def calc_Qest(C1:node, graph:Graph[Any, String]) {
        val r: Double = 0.25
        var Oinew: ListBuffer[VertexId] = new ListBuffer[VertexId]()
        var k:Int = 0
        var ouli:Double = 0
        var oul_min:Double = 999
//    if (result.nonEmpty) {
        if (false) {
          breakable { for (c <- result)  //c.O is Oresult
          {
            for (layer <- C1.So)
            {
              //Ei(O u candO,i)
              for (v <- C1.O) {
                Oinew += v
              }
              for (cand_temp <- C1.cand(layer.toInt - 1)) {
                if (!Oinew.contains(cand_temp)) {
                  Oinew += cand_temp
                }
              }
              Oinew = Oinew.sortWith(_ < _)
              k = calc_k(Oinew, c.O, graph, layer)
              ouli = calc_ouli(C1.O, c.O, graph, k, layer)
              if (ouli < oul_min) {
                oul_min = ouli
              }
              Oinew.clear()
            }
            //oul_min found < 0.5 no need to search further
            if (oul_min < 0.25) { break() }
          } }
        }
        else { oul_min = 0 }
        var γimax:Double = 0
        var min_degGi:Double = 0
        var nimax:Double = 0
        var γimax_List:ListBuffer[Double] = new ListBuffer[Double]
        var nimax_List:ListBuffer[Double] = new ListBuffer[Double]
        var max_List:ListBuffer[Double] = new ListBuffer[Double]
        var sum:Double = 0
        if (oul_min >= r)
          {
            C1.Q = -1
            println("oul_min = " + oul_min + " >= r => Subtree is redundant!")
```

```
            }
        else
            {
                for (layer <- C1.So){
                    for (v <- C1.O) {
                        Oinew += v
                    }
                    for (cand_temp <- C1.cand(layer.toInt - 1)) {
                        if (!Oinew.contains(cand_temp)) {
                            Oinew += cand_temp
                        }
                    }
                    min_degGi = find_min_degGi(C1.O, Oinew, graph, layer)
                    if ((min_degGi / C1.O.length) < 1) { γimax = min_degGi / C1.O.length }
                    else { γimax = 1 }
                    γimax_List += γimax
                    nimax = find_nimax(min_degGi, Oinew, graph, layer)
                    nimax_List += nimax
                }
                γimax_List = γimax_List.sortWith(_ > _)
                nimax_List = nimax_List.sortWith(_ > _)
                for (k <- 0 until C1.So.length)
                {
                        sum = 0
                        for (m <- 0 to k)
                        {
                            sum += γimax_List(m)
                        }
                        max_List += nimax_List(k) * sum
                }
                C1.Q = max_List.max
                if ((nimax_List.max < O_threshold) || (C1.So.length < S_threshold))
                    {
                        C1.Q = -1
                    }
            }
    }
        def calc_k(Oi:ListBuffer[VertexId], Oresult:ListBuffer[VertexId], graph:Graph[Any, String], layer:String): Int ={
            var k:Int = 0
            var minus_k:Int = 0
            val temp_subgraph1: Graph[Any, String] = graph.subgraph(vpred = (v, d) => Oi.contains(v), epred = e => layer ==
e.attr)
            val temp_subgraph2: Graph[Any, String] = graph.subgraph(vpred = (v,d) => Oi.contains(v) && Oresult.contains(v),
epred = e => layer == e.attr)
            if (result.isEmpty)
            {
                k = temp_subgraph1.numEdges.toInt
            }
            else
            {
//      for (triplet1 <- temp_subgraph1.triplets.collect ; triplet2 <- temp_subgraph2.triplets.collect)
                for (edges1 <- temp_subgraph1.edges.collect ; edges2 <- temp_subgraph2.edges.collect)
                {
                  if (edges1.equals(edges2))
                  {
                      minus_k +=1
                  }
            }
                k = temp_subgraph1.numEdges.toInt - minus_k
            }
            k
        }
        def calc_ouli(Oi:ListBuffer[VertexId], Oresult:ListBuffer[VertexId], graph:Graph[Any, String], k:Int, layer:String):
Double ={
            val temp_subgraph2: Graph[Any, String] = graph.subgraph(vpred = (v,d) => Oi.contains(v) && Oresult.contains(v),
epred = e => layer == e.attr)
            var ouli:Double = 0
```

```
        var temp:Double=0
        var max1:Double = 0
        val Ei_oul = temp_subgraph2.numEdges
        temp = 1/4 * (Oi.length^2 + Oi.length) - Ei_oul - k
        if (temp > 0) { max1 = temp }
        else { max1 = 0 }
        ouli = (Ei_oul + max1) / (Ei_oul + k + max1)
        ouli
      }
  def find_min_degGi(O:ListBuffer[VertexId], Oinew:ListBuffer[VertexId], graph:Graph[Any, String], layer:String):
Double ={
    val temp_subgraph2: Graph[Any, String] = graph.subgraph(vpred = (v,d) => Oinew.contains(v), epred = e => layer ==
e.attr)
    //find min degree
    var min_degree = 999
    var min_vertex: VertexId = 999
    val degrees: VertexRDD[Int] = temp_subgraph2.degrees
    if (O.length <= 1)
    {
      min_degree = 0
      min_vertex = 0
    }
    else {
      degrees.filter { case (vertex, deg) => min_degree >= deg }.collect()
        .foreach { case (vertex, deg) => if ((min_degree >= deg) && O.contains(vertex)) {
          min_degree = deg
          min_vertex = vertex
        }
        }
    }
    min_degree
  }
  def find_nimax(min_degGi:Double, Oinew:ListBuffer[VertexId], graph:Graph[Any, String], layer:String): Double ={
    var nimax:Double = 0
    var temp1:Double = 0
    var temp2:Double = 0
    temp1 = math.floor(min_degGi/0.5) + 1
    temp2 = graph.subgraph(vpred = (v,d) => Oinew.contains(v), epred = e => layer == e.attr).numVertices
    if ((temp1 > temp2)||(min_degGi == 0)) { nimax = temp2}
    else { nimax = temp1 }
    nimax
  }
  def prune1(C:node, layer:String): Unit = {
      for (i <- C.cand(layer.toInt - 1)) {
        var num = 0
        for (check_layer <- C.So) {
          if (C.cand(check_layer.toInt - 1).contains(i)) {
            num += 1
          }
        }
        if (num < S_threshold)
        {
          for (rem_layer <- C.So) {
            if (C.cand(rem_layer.toInt - 1).contains(i)) {
              C.cand(rem_layer.toInt - 1) -= i
            }
          }
        }
      }
  }
  def prune2(O:ListBuffer[VertexId], layer:String, OcandO:ListBuffer[VertexId], graph:Graph[Any, String]): Boolean = {
    var toprune = false
    val prunedegrees: VertexRDD[Int] = graph.subgraph(vpred = (v, d) => OcandO.contains(v), epred = e => layer ==
e.attr).degrees
    prunedegrees.filter { case (vertex, deg) => (deg < 4) && O.contains(vertex) }.collect()
      .foreach { case (vertex, deg) => if ((deg < 4) && O.contains(vertex)) {
```

```
            println("Layer "+ layer + " from subtree " + O + " must be pruned because it contains vertex " + vertex +"
that has " + deg + " < 4")
            toprune = true
          }
          }
    toprune
    }

    def prune_ceil(C:node, layer:String, OcandO:ListBuffer[VertexId], graph:Graph[Any, String]): Boolean = {
      var continue = false
      val prunedegrees: VertexRDD[Int] = graph.subgraph(vpred = (v, d) => OcandO.contains(v), epred = e => layer ==
e.attr).degrees
      prunedegrees.filter { case (vertex, deg) => deg < 4 }.collect()
        .foreach { case (vertex, deg) => if (deg < 4) {
          C.cand(layer.toInt - 1) -= vertex
          continue = true
        }
        }
      continue
    }
}
```

---

11. Βιβλιογραφία

[1]. http://spark.apache.org/

[2]. Nahoom-Kabakov, L. (2016). *Setting UP Spark 2.0 environment on intellij community edition version 2016.2.2*

[3]. Guanrong Chen, Xiaofan Wang, Xiang Li, (2015). *Fundamentals of Complex Networks: Models, Structures and Dynamics*. Higher Education Press.

[4]. Jungeun Kim, Jae-Gil Lee (2015). *Community Detection in Multi-Layer Graphs: A Survey*. Department of Knowledge Service Engineering, KAIST, Daejeon, Republic of Korea

[5]. Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl (2012). *Mining Coherent Subgraphs in Multi-Layer Graphs with Edge Labels*. Data Management and Data Exploration Group RWTH Aachen University, Germany

[6]. S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, Feb. 2010.

[7]. S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, Aug. 2007.

[8]. G. Liu and L. Wong. *Effective pruning techniques for mining quasi-cliques*. In: ECML/PKDD (2), pages 33–49, 2008.

[9]. J. Pei, D. Jiang, and A. Zhang. *On mining cross-graph quasi-cliques*. In SIGKDD, pages 228–238, 2005.

[10]. Z. Zeng, J. Wang, L. Zhou, and G. Karypis. *Coherent closed quasi-clique discovery from large dense graph databases*. In SIGKDD, pages 797–802, 2006.

[11]. Guimei Liu and Limsoon Wong. *Effective Pruning Techniques for Mining Quasi-cliques*. School of Computing, National University of Singapore, Singapore, 2008

[12]. R. Rymon. *Search through systematic set enumeration*. In KR, pages 539–550, 1992.

[13]. C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers: *Biogrid: a general repository for interaction datasets*. In: Nucleic Acids Research 2006 34 (1) D535–D539

[14]. M. De Domenico, V. Nicosia, A. Arenas, and V. Latora: *Structural reducibility of multilayer networks*. In: Nature Communications 2015 6, 6864

[15]. Manlio De Domenico, Andrea Lancichinetti, Alex Arenas, and Martin Rosvall: *Identifying Modular Flows on Multilayer Networks Reveals Highly Overlapping Organization in Interconnected Systems*. In: Physical Review X 5, 011027 (2015)

[16]. Pei, J., Jiang, D., Zhang, A.: *On mining cross-graph quasi-cliques*. In: Proc. of the 11th ACM SIGKDD Conference. (2005) 228–238.

[17]. F. Moser, R. Colak, A. Rafiey, and M. Ester, *Mining cohesive patterns from graphs with feature vectors* in SDM, 2009, pp. 593–604.

[18]. S. G*ü*nnemann, I. F¨arber, B. Boden, and T. Seidl. *Subspace clustering meets dense subgraph mining: A synthesis of two paradigms*. In *ICDM*, pages 845–850, 2010.

Πρόσθετες Πηγές:

[19]. Παπαδόπουλος, Α. (2015). Διάλεξη: *An Introduction to Scala*

[20]. Παπαδόπουλος, Α. (2015). Διάλεξη: *An Introduction to Spark*