ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ
ΤΟΜΕΑΣ ΜΑΘΗΜΑΤΙΚΩΝ

ΑΝΑΛΥΣΗ ΚΟΙΝΩΝΙΚΩΝ ΔΙΚΤΥΩΝ, ΥΠΟΛΟΓΙΣΤΙΚΕΣ ΜΕΘΟΔΟΙ ΚΑΙ ΜΕΘΟΔΟΙ ΑΠΕΙΚΟΝΙΣΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΣΕΤΤΑΣ-ΚΑΦΦΕΤΖΗΣ ΑΛΕΞΙΟΣ

Επιβλέπων: Αντώνιος Συμβώνης
Καθηγητής ΕΜΠ

Αθήνα Μάρτιος 2012

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας αποτελεί σε πρώτο σκέλος η ανάπτυξη μιας εφαρμογής για την Ανάλυση Κοινωνικών Δικτύων(Social Network Analysis,SNA). Η λειτουργία της εφαρμογής αυτής στηρίζεται στη δημιουργία του προσωπικού δικτύου ενός χρήστη, στον υπολογισμό των μέτρων που προκύπτουν από την θεωρία των Κοινωνικών Δικτύων για το δίκτυο αυτό και την απεικόνιση του με την μορφή ενός κατευθυνόμενου γραφήματος. Η εφαρμογή αυτή παρέχει επίσης αρκετές δυνατότητες παραμετροποίησης του τελικού αποτελέσματος, δηλαδή της απεικόνισης του δικτύου αυτού. Στην συνέχεια γίνεται η προσπάθεια ερμηνείας τον αποτελεσμάτων της παραπάνω εφαρμογής, δηλαδή των διαφορετικών απεικονίσεων που προκύπτουν ανάλογα με την οπτική γωνία που εξετάζουμε το δίκτυο, οι οποίες προκύπτουν ανάλογα με το μέτρο της Ανάλυσης Κοινωνικών Δικτύων που έχουμε επιλέξει.

Το έναυσμα για την διπλωματική εργασία μου δόθηκε στην εταιρεία όπου έκανα πρακτική, Datamine, η οποία δραστηριοποιείται στον τομέα της εξόρυξης δεδομένων (dataming) και μία αναπτυσσόμενη περιοχή σε αυτό αποτελεί η Ανάλυση Κοινωνικών Δικτύων. Τα δεδομένα που χρησιμοποίησα μου τα παρείχαν από την εταιρεία και αποτελούν τεχνητά δεδομένα με την μορφή μίας βάσης δεδομένων τηλεπικοινωνιακών συνδιαλέξεων. Εκτός από τον τομέα των Τηλεπικοινωνιών, η Ανάλυση Κοινωνικών Δικτύων μπορεί να εφαρμοστεί σε αρκετούς ακόμα τομείς, όπως τα διαδικτυακά κοινωνικά δίκτυα, οι συναλλαγές μεταξύ πελατών μίας τράπεζας, οι διατραπεζικές συναλλαγές, τα δίκτυα τρομοκρατίας και γενικά σε όσα δίκτυα υπάρχει αλληλεπίδραση μεταξύ των μελών τους.

Γενικά με την Ανάλυση Κοινωνικών Δικτύων μπορεί να επιτευχθεί η απομόνωση ορισμένων «ισχυρών» κόμβων ή κόμβων που εμφανίζουν μια συγκεκριμένη δραστηριότητα. Ειδικά για δίκτυα με μεγάλο μέγεθος, δηλαδή με αρκετούς χρήστες, η σωστή απεικόνιση τους και ο υπολογισμός των κατάλληλων μέτρων αποτελεί σημαντικό εργαλείο για την εξαγωγή συμπερασμάτων. Λόγω του μεγάλου όγκου των σύγχρονων βάσεων δεδομένων και την ανάγκη για γρήγορα και σαφή αποτελέσματα η Ανάλυση Κοινωνικών Δικτύων αποτελεί ένα χρήσιμο εργαλείο προς την κατεύθυνση αυτή.

Περιεχόμενα

Περίληψη **2**

Περιεχόμενα **3**

Αντικείμενο Εργασίας **4**

Θεωρητικό Μέρος **5**

Κοινωνικά Δίκτυα *5*

Θεωρία Γραφημάτων *6*

Γλώσσες Προγραμματισμού *8*

Μέτρα Ανάλυσης Κοινωνικών Δικτύων *9*

Σχεδίαση Γραφημάτων *21*

Περιγραφή Εφαρμογής **23**

Ανάκτηση Δεδομένων *24*

Υπολογισμός Μέτρων *25*

Απεικόνιση Γραφήματος *27*

Ανάλυση Δικτύων **29**

Επεκτάσεις Εφαρμογής **36**

Αποτίμηση **38**

Βιβλιογραφία **39**

Παράρτημα **40**

Αντικείμενο Εργασίας

Το κυριότερο μέρος της διπλωματικής εργασίας αποτέλεσε η δημιουργία και ο σχεδιασμός της εφαρμογής που περιγράφηκε περιληπτικά παραπάνω, στη παρούσα εργασία θα αποφευχθεί όμως η εκτενής ανάλυση του κώδικα και των μεθόδων που απαρτίζουν την εφαρμογή αυτή. Αντί αυτού ο κώδικας και η αρχιτεκτονική της εφαρμογής παρουσιάζονται στο Παράρτημα για επεξήγηση οποιονδήποτε απορριών . Ο τρόπος λειτουργίας της παρουσιάζεται αρκετά αναλυτικά στο αντίστοιχο κεφάλαιο και βασίζεται σε τρία μέρη.

Στο πρώτο μέρος επιλέγω ένα κόμβο από τη βάση δεδομένων που δίνεται και χτίζω το δικό του προσωπικό δίκτυο ανάλογα με το επίπεδο γειτνίασης που μας ενδιαφέρει. Στο δεύτερο μέρος υπολογίζονται τα επιλεγμένα από τη Βιβλιογραφία μέτρα της Ανάλυσης Κοινωνικών Δικτύων (SNA) που χαρακτηρίζουν τον κάθε κόμβο. Ενώ στο τρίτο μέρος πραγματοποιείται η οπτικοποίηση του παραπάνω δικτύου, για την οπτικοποίηση αυτή δίνονται διαφορετικές επιλογές σύμφωνα με το αποτέλεσμα που επιθυμούμε. Πάνω στο δίκτυο που έχει προκύψει μας υπάρχουν διαδραστικές επιλογές που βοηθούν στην καλύτερη επεξεργασία του δικτύου. Επίσης προτείνονται ορισμένα περαιτέρω μέτρα SNA(Social Network Analysis) και επεκτάσεις της εφαρμογής αυτής ανάλογα τις ανάγκες που υπάρχουν.

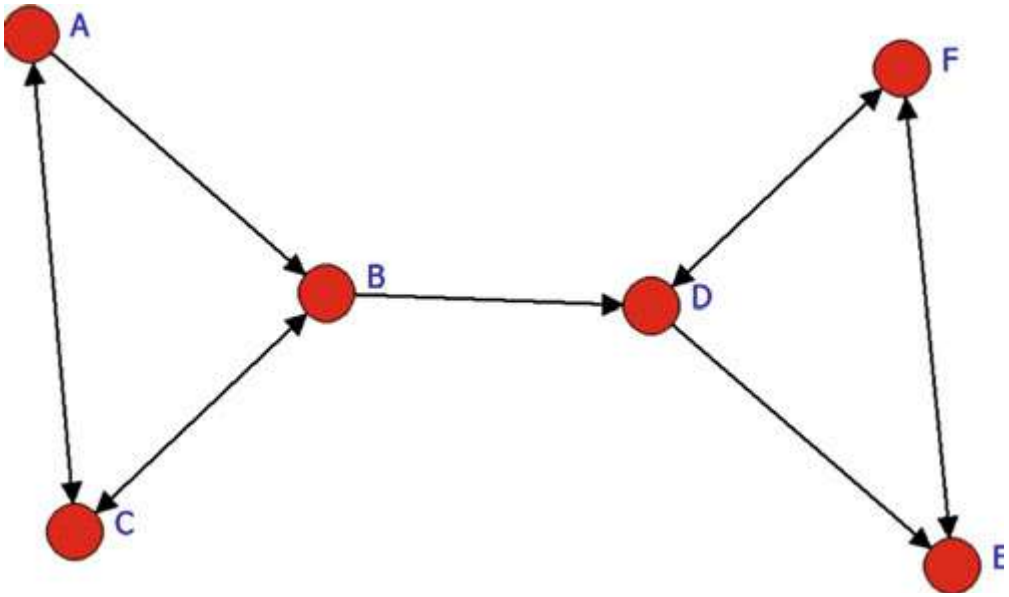
Στο τέλος της διπλωματικής μου εργασίας γίνεται η προσπάθεια ερμηνείας και ανάλυσης ορισμένων χαρακτηριστικών απεικονίσεων που προκύπτουν από επιλεγμένα δίκτυα. Επεξηγούνται ,επίσης, με βάση τις απεικονίσεις αυτές τα μέτρα της Ανάλυσης Κοινωνικών Δικτύων που επέλεξα από τη Βιβλιογραφία.

Η εργασία αυτή εκτός από τον τομέα της Ανάλυσης Κοινωνικών Δικτύων, δανείζεται στοιχεία από τη Θεωρία Γραφημάτων, τη Θεωρία Απεικόνισης Γραφημάτων, την επεξεργασία και την χρήση Βάσεων Δεδομένων (μέσω της SQL) και τον αντικειμενοστραφή προγραμματισμό με τη χρήση της γλώσσας C#. Τα παραπάνω αποτελούν το Θεωρητικό μέρος της εργασίας και αναλύονται στο πρώτο μέρος της εργασίας.

Θεωρητικό Μέρος

Κοινωνικά Δίκτυα

Ένα κοινωνικό δίκτυο ορίζεται ως ένα σύνολο κοινωνικών ατόμων, κόμβων(nodes) ή μελών τα οποία είναι συνδεδεμένα με ένα ή περισσότερα είδη σχέσεων(ties). Παραδείγματα τέτοιων δικτύων αποτελούν οι πελάτες μια τηλεπικοινωνιακής εταιρείας, τα μέλη ενός διαδικτυακού κοινωνικού δικτύου, οι πελάτες μία τράπεζας ακόμη και τα μέλη μία τρομοκρατικής οργάνωσης. Για την κατανόηση των δικτύων και των συμμετεχόντων τους, οι ερευνητές πρέπει να προσδιορίσουν τη θέση των ατόμων μέσα στο δίκτυο. Για το λόγο αυτό χρησιμοποιούν τα εργαλεία της θεωρίας γραφημάτων. Ο προσδιορισμός της θέσης κάθε κόμβου μπορεί να επιτευχθεί με την έννοια της κεντρικότητας(centrality) και άλλων συναφών εννοιών. Με τα εργαλεία αυτά μπορούμε να ανακαλύψουμε διάφορους ρόλους των κόμβων μέσα στο δίκτυο, όπως οι ηγέτες(leaders), οι γέφυρες(bridges), οι απομονωμένοι(isolates) και άλλα. Για τα παραπάνω βασικά χαρακτηριστικά στοιχεία ενός κοινωνικού δικτύου και της θεωρίας γραφημάτων ακολουθεί περαιτέρω ανάλυση.(Scott, 1987)



Παράδειγμα Προσανατολισμένου Γραφήματος. Γράφημα 1.

Θεωρία Γραφημάτων

Ακμές (Ties)

Οι ακμές ή οι πλευρές συνδέουν δύο ή περισσότερους κόμβους σε ένα γράφημα. Πολλές ανθρώπινες συμπεριφορές όπως η αναζήτηση πληροφοριών, ο δανεισμός χρημάτων αποτελούν κατευθυνόμενες ακμές (directedties), ενώ οι συμμαχίες είναι μη-κατευθυνόμενες ακμές (undirectedties). Οι κατευθυνόμενες ακμές μπορεί να είναι ανταποδοτικές, στην περίπτωση που ένας κόμβος επισκέπτεται έναν άλλο και το αντίστροφο, και μπορεί να υπάρχουν μόνο προς μία κατεύθυνση όταν μόνο ο ένας επισκέπτεται τον άλλο. Τόσο οι κατευθυνόμενες όσο και οι μη-κατευθυνόμενες ακμές θεωρούνται δυαδικές, δηλαδή είτε υπάρχουν ή όχι. Επιπλέον υπάρχουν οι ακμές με τιμές (το βάρος τους,weight), ανάλογα με το αν είναι ισχυρές ή ασθενής, δηλαδή αν μεταφέρουν αρκετά ή λιγότερα δεδομένα και άλλα.

Για την ύπαρξη μίας ακμής ανάμεσα σε δύο κόμβους i,j μπορούμε να ορίσουμε την μεταβλητή:

$$x_{ij} = \begin{cases} 1, \alpha\nu\ \upsilon\pi\acute{\alpha}\rho\chi\epsilon\iota\ \sigma\acute{\upsilon}\nu\delta\epsilon\sigma\eta\ \alpha\pi\acute{o}\ \tau o\nu\ \kappa\acute{o}\mu\beta o\ i\ \sigma\tau o\nu\ j \\ 0, \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases}$$

Με τη βοήθεια της μεταβλητής αυτής μπορούμε να δημιουργήσουμε τον πίνακα επικοινωνίας, όπου θα αναπαριστά ποιοι κόμβοι επικοινωνούν μεταξύ τους. Σε ένα δίκτυο g κόμβων ο πίνακας αυτός θα είναι μεγέθους g x g, όπου κάθε γραμμή και κάθε στήλη θα αποτελούν τους κόμβους και τα κελία (i,j) θα παίρνουν τις τιμές της x_{ij}.

Πίνακας Επικοινωνίας Γραφήματος 1

Κόμβοι	**A**	**B**	**C**	**D**	**E**	**F**
A	0	1	1	0	0	0
B	0	0	1	1	0	0
C	1	1	0	0	0	0
D	0	0	0	0	1	1
E	0	0	0	0	0	1
F	0	0	0	1	1	0

Εξωτερικός Βαθμός(Out-Degree)

Ο εξωτερικός βαθμός ενός κόμβου αποτελεί τον αριθμό των ακμών, σε ένα κατευθυνόμενο γράφημα, που ξεκινάν από τον κόμβο και κατευθύνονται προς άλλους κόμβους του δικτύου. Στο γράφημα 1 ο εξωτερικός βαθμός του κόμβου A είναι 2, όπως επίσης και του B και του C.

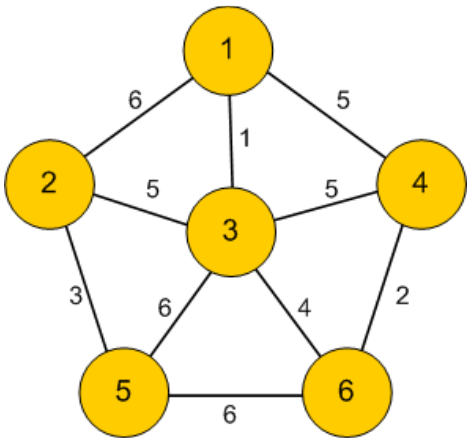
Εσωτερικός Βαθμός(In-Degree)

Ο εσωτερικός βαθμός ενός κόμβου αποτελεί τον αριθμό των ακμών, σε ένα κατευθυνόμενο γράφημα, που κατευθύνονται προς τον κόμβο και ξεκινάν από άλλους κόμβους του δικτύου. Στο γράφημα 1 ο εσωτερικός βαθμός του κόμβου A είναι 1, ενώ του B και του C είναι 2 .

Βάρος Ακμής (Tie weight)

Το βάρος μίας ακμής είναι το κόστος που έχει η σύνδεση δύο κόμβων, αυτό θα μπορούσε να είναι η χιλιομετρική απόσταση ανάμεσα σε δύο πόλεις ή το κόστος του τηλεφωνήματος ανάμεσα σε δύο άτομα. Το βάρος συμβολίζεται ως w_{ij} και ορίζεται ως το κόστος της σύνδεσης από τον κόμβο i στον κόμβο j.

Με βάση τη μεταβλητή του βάρους μπορούμε να δημιουργήσουμε τον αντίστοιχο gxg πίνακα όπως προηγουμένως όπου στα κελιά (i,j) θα έχει τα αντίστοιχα βάρη.
Στο παρακάτω γράφημα παρουσιάζεται ένα γράφημα με βάρη.



Γράφημα 2,Γράφημα με βάρη.

Πίνακας Επικοινωνίας Γραφήματος 2

Κόμβοι	**1**	**2**	**3**	**4**	**5**	**6**
1	0	6	1	5	0	0
2	6	0	5	0	3	0
3	1	5	0	5	6	4
4	5	0	5	0	0	2
5	0	3	6	0	0	6
6	0	0	4	2	6	0

Πυκνότητα (Density)

Μια από τις πιο κοινές έννοιες της θεωρίας γραφημάτων είναι αυτή της πυκνότητας, που περιγράφει το μέγεθος της σύνδεσης των κόμβων. Ένας πλήρης κόμβος είναι αυτός που συνδέεται άμεσα με όλους τους υπόλοιπους του γραφήματος. Η πυκνότητα ενός γραφήματος ορίζεται ως το πηλίκο των ακμών αυτού προς το συνολικό αριθμό των κόμβων του γραφήματος. Αποτελεί μία ένδειξη για το βαθμό συνεκτικότητας του γραφήματος.

Μονοπάτι (Path) ,Μήκος (Length) και Απόσταση (Distance)

Οι κόμβοι μπορεί να συνδέονται μεταξύ τους με μία ακμή ή μέσω μιας ακολουθίας ακμών. Η ακολουθία αυτή ονομάζεται δρόμος, και ένας δρόμος του οποίου κάθε κόμβος και κάθε ακμή είναι μοναδικά, ονομάζεται μονοπάτι(path). Το μονοπάτι, ύστερα από τον κόμβο και την ακμή, είναι μία από τις πιο βασικές έννοιες στη θεωρία γραφημάτων. Το μήκος (length) του μονοπατιού μετριέται από τον αριθμό των ακμών που αποτελείται. Η απόσταση (distance) ανάμεσα σε δύο κόμβους είναι το συντομότερο μονοπάτι (ονομάζεται και γεωδαιτική,geodesic) που τους συνδέει. (Παπαϊωάννου, 2006)

Κλίκες (Cliques)

Μία κλίκα σε ένα γράφημα, είναι ένα υπό-γράφημα στο οποίο κάθε κόμβος είναι άμεσα συνδεδεμένος με κάθε άλλο κόμβο του υπό-γραφήματος. Στο παραπάνω παράδειγμα (Γράφημα 1) υπάρχουν δύο κλίκες, η {A,B,C} και η {D,E,F}.

Γέφυρα (Bridge)

Γέφυρα ονομάζεται η ακμή που όταν διαγραφεί αυξάνει τα συνδεδεμένα μέρη (connectedcomponents) , δηλαδή τα μέρη που δεν επικοινωνούν με το υπόλοιπο γράφημα. Ένας εναλλακτικός ορισμός είναι ότι γέφυρα αποτελεί η ακμή αυτή που δεν ανήκει σε κανένα κύκλο (κλειστό μονοπάτι, δηλαδή έχει ίδια αρχή και τέλος. Με βάση την πλευρική συνεκτικότητα που ορίζεται ως ο ελάχιστος αριθμός πλευρών που πρέπει να αφαιρέσουμε από ένα γράφημα ώστε να προκύψει γράφημα μή συνεκτικό, αρκεί να έχει βαθμό 1 για να υπάρχει μία γέφυρα. (Παπαϊωάννου, 2006)

Γλώσσες Προγραμματισμού

SQL

Η γλώσσα SQL που συχνά αναφέρεται ως Structured Query Language, είναι μια υπολογιστική γλώσσα για βάσεις δεδομένων που σχεδιάστηκε για τη διαχείριση των στοιχείων σε συστήματα διαχείρισης δεδομένων. Το πεδίο εφαρμογής της περιλαμβάνει την εισαγωγή δεδομένων, ενημέρωση και διαγραφή αυτών, δημιουργία σχημάτων και τροποποίηση, των στοιχείων ελέγχου της βάσης δεδομένων. Η SQL ήταν μια από τις πρώτες εμπορικές γλώσσες και έγινε η πιο διαδεδομένη γλώσσα βάσης δεδομένων.(Steohens, etal., 2008)

C#

Η C #είναι μια απλή, μοντέρνα, γενικής χρήσης, αντικειμενοστραφής γλώσσα προγραμματισμού. Αναπτύχθηκε από τη Microsoft μέσα στην .NET και αργότερα εγκρίθηκε ως πρότυπο από Ecma(ECMA-334) και ISO(ISO /IEC23270).Η C# είναι μια από τις γλώσσες προγραμματισμού που έχουν σχεδιαστεί για την Common Language Infrastructure (CLI).

Η C# παρουσιάζει αρκετές ομοιότητες με την Java, ο σχεδιασμός της στηρίζεται στις κλάσεις και τις μεθόδους, ενώ υποστηρίζει πολύ καλά τη δημιουργία WindowsForms. Επιπλέον μέσω συγκεκριμένων libraries που διαθέτει είναι εύκολη η σύνδεση της με την SQL και η ανάκτηση, επεξεργασία των δεδομένων.(Griffiths, etal., 2010)

Μέτρα Ανάλυσης Κοινωνικών Δικτύων

Κεντρικότητα(Centrality)

Τα μέτρα της κεντρικότητας αναγνωρίζουν του προεξέχοντες κόμβους, ειδικά τα «αστέρια», δηλαδή αυτούς που συσχετίζονται πιο ενεργά με τα υπόλοιπα μέλη του δικτύου. Τα πιο σημαντικά μέτρα της κεντρικότητας είναι ο βαθμός κεντρικότητας (degreecentrality),η παραπλήσια κεντρικότητα (closenesscentrality) , η between-nesscentrality, η κεντρικότητα πληροφορίας (informationcentrality) και η κεντρικότητα ιδιοδιανυσμάτων (eigenvectorcentrality) (Wasserman, et al., 1994)

1. Ο βαθμός κεντρικότητας είναι το άθροισμα όλων των κόμβων που είναι άμεσα συνδεδεμένοι με ένα κόμβο, δείχνει δραστηριότητα και διασημότητα. Όσο περισσότερες ακμές φεύγουν από ένα κόμβο τόσο αυξάνει ο βαθμός κεντρικότητας του.
2. Η παραπλήσια κεντρικότητα βασίζεται στην έννοια της απόστασης. Αν ένας κόμβος είναι κοντά σε όλους του άλλους του δικτύου, δηλαδή έχει απόσταση ίση με ένα, τότε είναι ανεξάρτητος και δεν χρειάζεται κανέναν άλλο κόμβο για να συνδεθεί με όλο το δίκτυο. Η παραπλήσια κεντρικότητα μετράει ανεξαρτησία και αποδοτικότητα.
3. Between-nesscentrality, αυτή η κεντρικότητα είναι ο αριθμός των ζευγαριών που συνδέει ένας κόμβος, τα οποία χωρίς αυτών δεν θα συνδέονταν. Αποτελεί ένα μέτρο ελέγχου, γιατί ένας κόμβος με υψηλή Between-nesscentrality μπορεί να λειτουργήσει ως «φύλακας» για τους κόμβους που συνδέει και ελέγχει τη ροή των δεδομένων ή της πληροφορίας ανάμεσα στους κόμβους αυτούς.
4. Η κεντρικότητα πληροφορίας έχει παρόμοιο σκοπό με αυτόν της Between-ness, με τη διαφορά του ότι λαμβάνει υπόψη τα βάρη των ακμών στους υπολογισμούς. Στην ουσία επικεντρώνεται στο μέγεθος της πληροφορίας που υπάρχει σε κάθε ακμή.
5. Η κεντρικότητα ιδιοδιανυσμάτων είναι ένα μέτρο της σημαντικότητας ενός κόμβου μέσα σε ένα δίκτυο, βασίζεται στην αρχή ότι η σύνδεση σε κόμβους μεγάλης κεντρικότητας ωφελούν περισσότερο τον κόμβο. Στην ουσία είναι μια βελτιωμένη μορφή του βαθμού κεντρικότητας.

Κεντρικότητα Γραφήματος (Group Centrality)

Τα παραπάνω μέτρα μπορούν να γενικευτούν και για το σύνολο του γραφήματος, όπου μπορούμε να κατανοήσουμε την ετερογένεια του γραφήματος με βάση την κεντρικότητα των κόμβων του. Η κεντρικότητα ενός γραφήματος μπορεί να θεωρηθεί σαν ένα μέτρο μεταβλητότητας, διασποράς ή εξάπλωσης για τη σύγκριση ανάμεσα σε γραφήματα.

Πρεστίζ (Prestige)

Η διαφορά μεταξύ της κεντρικότητας και του πρεστίζ έγκειται στο γεγονός ότι για το πρεστίζ μας αφορούν μόνο όσες ακμές κατευθύνονται προς τον κόμβο και όχι όσες φεύγουν από αυτό, άρα επικεντρωνόμαστε στον κόμβο σαν αποδέκτη. Γενικότερα με το πρεστίζ δείχνει πόσο μπορεί να επηρεάσει ένας κόμβο το δίκτυο.

Βαθμός Κεντρικότητας (Degree Centrality)

Κόμβου

Ο βαθμός ενός κόμβου είναι το πρώτο μέτρο που θα ασχοληθούμε και αφορά τον αριθμό των κόμβων που συνδέεται αυτός και ορίζεται από το άθροισμα τους, δηλαδή:
$C_D(n_i) = d(n_i) = \sum_j x_{ij} = k_i$.

Ενώ για μεγαλύτερη ευχέρεια στη σύγκριση μεταξύ των βαθμών ενός κόμβο μπορούμε να χρησιμοποιήσουμε και τον κανονοικοποιημένο βαθμό κόμβου(διαιρούμε το βαθμό με το σύνολο των δυνατών συνδέσεων, δηλαδή τους υπόλοιπους g-1 κόμβους):
$C'_D(n_i) = \frac{d(n_i)}{g-1}$,
όπου g είναι ο αριθμός των κόμβων και παίρνει τιμές από 0 έως 1.

Για παράδειγμα στο Γράφημα 1, έχουμε

Βαθμοί Κεντρικότητας Κόμβων

n_i	$C_D(n_i)$	$C'_D(n_i)$
A	2	0.4
B	2	0.4
C	2	0.4
D	2	0.4
E	1	0.2
F	2	0.4

Ο βαθμός ενός κόμβου σε ένα γράφημα με βάρη έχει διαφορετικό ορισμό, ονομάζεται η δύναμη του κόμβου(nodestrength) και ορίζεται ως εξής:

$C_D^w(n_i) = \sum_{j=1}^{g} wij = s_i$.
Έτσι για το Γράφημα 2 έχουμε:

Βαθμοί Κεντρικότητας Κόμβων με βάρη

n_i	$C_D^w(n_i)$
1	12
2	14
3	21
4	12
5	15
6	12

Ένας συνδυασμός των δύο παραπάνω βαθμών προτείνετε από τους T.Opsahl ,F.Agneessens, J.Skvoretz και είναι ο εξής : (Opsahl, et al., 2010)
$C_D^{w\alpha}(n_i) = k_i^{1-\alpha} * s_i^{a}$,
όπου το α είναι μια παράμετρος που καθορίζεται από τα δεδομένα με βάση το που θέλουμε να δώσουμε έμφαση. Δηλαδή αν έχει τιμή από 0 έως 1 τότε δίνουμε πλεονέκτημα στους

κόμβους με μεγαλύτερο βαθμό κεντρικότητας, ενώ εάν η τιμή του α είναι πάνω από 1 πλεονέκτημα έχει ένας κόμβος με μεγαλύτερη δύναμη.

Με το καινούριο αυτό μέτρο έχουμε στο γράφημα 2.

Πίνακας Βαθμών Κεντρικότητας

			$C_D^{w\alpha}(n_i)$			
n_i	$C_D(n_i)$	$C_D^w(n_i)$	**α=0**	**α=0.5**	**α=1**	**α=2**
1	3	12	3	6	12	48
2	3	14	3	7.21	14	65.33
3	5	21	5	10.25	21	88.2
4	3	12	3	6	12	48
5	3	15	3	6.71	15	75
6	3	12	3	6	12	48

Γραφήματος

Ο βαθμός αυτός αποτελεί ένα μέτρο σύγκρισης ανάμεσα στα γραφήματα και αφορά κατά πόσο ένα γράφημα είναι κεντρικό. Για τον υπολογισμό αυτού χρησιμοποιούμε τη διασπορά του βαθμού των κόμβων και ένα μέτρο που πρότεινε ο Freeman (1979)
Η διασπορά ορίζεται ως εξής:

$S_D^2 = \frac{\sum_{i=1}^{g}(C_D(n_i)-\overline{C_D})^2}{g}$, όπου $\overline{C_D} = \frac{\sum_{i=1}^{g} C_D(n_i)}{g}$.

Και το μέτρο του Freeman ως εξής:
$C_D = \frac{\sum_{i=1}^{g}[C_D(n^*)-C_D(n_i)]}{[(g-1)(g-2)]}$, όπου $C_D(n^*) = \max_i C_D(n_i)$.
Έτσι για το Γράφημα 1 έχω:
$\overline{C_D} = 1.83$ και $C_D(n^*) = 2$, συνεπώς από τον τύπο της διασποράς $C_D = 0.1389$ και από τον τύπο του Freeman$C_D = 0.05$.

Ένα ακόμα μέτρο σύγκρισης των γραφημάτων είναι η πυκνότητα τους, που ορίζεται ως εξής:

$\Delta = \frac{\sum_{t=1}^{g} C_D'(n_i)}{g}$,

που στο παράδειγμα μας είναι Δ=0.183.

Παραπλήσια Κεντρικότητα (Closeness Centrality)

Κόμβου

Το δεύτερο αυτό είδος της κεντρικότητας αφορά το πόσο κοντά είναι ένας κόμβος σε ένα άλλο και προσδιορίζει το πόσο γρήγορα μπορεί να επικοινωνήσει ένας κόμβος με τους υπόλοιπους του δικτύου. Οι κόμβοι με υψηλή παραπλήσια κεντρικότητα είναι δυνατόν να μεταφέρουν γρήγορα πληροφορίες σε όλο το δίκτυο. Για τον υπολογισμό του μέτρου της παραπλήσιας κεντρικότητας ορίζουμε την γεωδαιτική $d(n_i, n_j)$ με i≠j,ως το ελάχιστο μονοπάτι ανάμεσα στους κόμβους i και j. Για τα δίκτυα με κατευθυνόμενους κόμβους έχουμε ότι $d(n_i, n_j) \neq d(n_j, n_i)$ γιατί οι γεωδαιτικές μπορεί να διαφέρουν. Έτσι προκύπτει για το βαθμό της παραπλήσιας κεντρικότητας:

$C_C(n_i) = \left[\sum_{j=1}^{g} d(n_i, n_j)\right]^{-1}$,

Ενώ για το κανονικοποιημένο βαθμό παραπλήσιας κεντρικότητας έχουμε:

$C'_C(n_i) = \frac{g-1}{\sum_{j=1}^{g} d(n_i,n_j)} = (g-1) * C_C(n_i)$, με τιμές από 0 έως 1.

Το μεγάλο μειονέκτημα του τύπου αυτού είναι πως για απομακρυσμένους κόμβους που έχουν γεωδαιτική 0, το άθροισμα αυτό γίνεται άπειρο, που υπολογιστικά δημιουργεί πρόβλημα. Επιπλέον στα κατευθυνόμενα δίκτυα υπάρχουν κόμβοι που δεν επικοινωνούν με όλους του δικτύου. Για το λόγο αυτό πρέπει να προσαρμόσουμε τον παραπάνω τύπο στα κατευθυνόμενα δίκτυα(όπως αυτό των τηλεπικοινωνιών) εισάγοντας μία νέα μεταβλητή, το πεδίο επιρροής (influence range) ενός κόμβου J_i. ΤοJ_i ορίζεται ως το σύνολο των κόμβων που επικοινωνεί ο κόμβος n_i, δηλαδή όσους κόμβους μπορεί να υπάρξουν μονοπάτια με τον n_i και ο τύπος της παραπλήσιας κεντρικότητας μετατρέπεται:

$$C_C^*(n_i) = \frac{\frac{J_i}{g-1}}{\sum_{j=1}^{g} \frac{d(n_i,n_j)}{J_i}} .$$

Βαθμοί Παραπλήσιας Κεντρικότητας Κόμβων

n_i	$C_C^*(n_i)$
A	0.1
B	0.125
C	0.1
D	0.4
E	0.267
F	0.4

Το πρόβλημα που υπάρχει με τους παραπάνω τύπους είναι ο υπολογισμός των γεωδαιτικών και η επιλογή του κατάλληλου αλγορίθμου για το σκοπό αυτό. Ενώ για τον υπολογισμό αυτό υπάρχουν δύο περιπτώσεις, γράφημα με βάρη και γράφημα χωρίς βάρη.

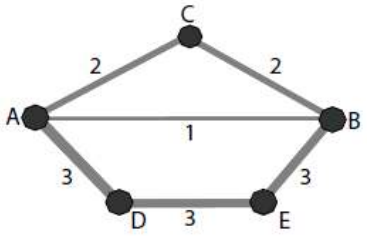
Στη περίπτωση του τηλεπικοινωνιακού δικτύου έχουμε γράφημα με κατευθυνόμενα βάρη και έτσι θα αναλύσουμε ένα αλγόριθμο για αυτή την περίπτωση.
Υπάρχουν πολλοί αλγόριθμοι για τον παραπάνω υπολογισμό. Ένας προτεινόμενος είναι αυτός του Dijkstra για την εύρεση της συντομότερης διαδρομής από ένα κόμβο. (Kleinberg, και συν., 2006)

Για την περίπτωση που έχουμε βάρη στο γράφημα, έχουν προτείνει ένα εναλλακτικό υπολογισμό οι T.Opsahl ,F.Agneessens, J.Skvoretz , όπου η γεωδαιτική ορίζεται από τα βάρη ως εξής:

$$d^{w\alpha}(n_i, n_j) = \min\left(\frac{1}{(w_{ih})^{\alpha}} + \cdots + \frac{1}{(w_{hj})^{\alpha}}\right),$$

δηλαδή ως το άθροισμα των αντιστρόφων το βαρών για το συντομότερο μονοπάτι. Ο λόγος που προτείνουν αυτό τον τύπο είναι πως ο αντίστροφος του βάρους προσδιορίζει πόσο «γρήγορα» μπορεί να κινηθείς. Ενώ η παράμετρος α προσδιορίζει αν θέλουμε να δώσουμε πλεονέκτημα στα μονοπάτια για α<1 και με α>1 δίνουμε πλεονέκτημα στο βάρος το μονοπατιών.

Στο παρακάτω γράφημα έχουμε :



			$d^{wa}(A,B)$			
μονοπάτι	***d(A,B)***	$d^{w}(A,B)$	**a=0**	**a=0.5**	**a=1**	**a=1.5**
{A,B}	1	1	1	1	1	1
{A,C,B}	2	1	2	1.4	1	0.7
{A,D,E,B}	3	1	3	1.8	1	0.5

Και η παραπλήσια κεντρικότητα ορίζεται:

$$C_C^{wa}(n_i) = \left[\sum_{j=1}^{g} d^{wa}(n_i, n_j)\right]^{-1} .$$

Γραφήματος

Το δεύτερο αυτό είδος της κεντρικότητας αφορά το πόσο κοντά είναι μεταξύ τους οι κόμβοι ενός γραφήματος και προσδιορίζει το πόσο γρήγορα μπορούν να επικοινωνήσουν. Ο πρώτος τύπος είναι του Freeman:

$$C_c = \frac{\sum_{i=1}^{g}[C_c'(n^*) - C_c'(n_i)]]}{\frac{[(g-2)(g-1)]}{2g-3}}$$,όπου το $C_c'(n^*) = \max_i C_c'(n_i)$.

Για παράδειγμα στο Γράφημα 1 είναι $C_c = 6.05$.

Επίσης μπορούμε να χρησιμοποιήσουμε και πάλι το στατιστικό τύπο της διασποράς για την παραπλήσια κεντρικότητα.

$$S_C^2 = \frac{\sum_{i=1}^{g}\left(C_C'(n_i) - \overline{C_C}\right)^2}{g}$$, όπου $\overline{C_C} = \frac{\sum_{i=1}^{g} C_C'(n_i)}{g}$.
Για το Γράφημα 1 είναι $\overline{C_C} = 2.76$ και το $S_C^2 = 3.76$.

Bettweenness Centrality

Κόμβος

Η επικοινωνία ορισμένων απομακρυσμένων κόμβων εξαρτάται από αυτούς που βρίσκονται ενδιάμεσα τους, έτσι όσοι κόμβοι ανήκουν σε αρκετές γεωδαιτικές είναι αυτοί που έχουν ένα είδος ελέγχου στο δίκτυο.

Αρχικά ορίζουμε τον αριθμό των γεωδαιτικών από τον κόμβο j στον κόμβο k ως g_{jk}. Ενώ για τον αριθμό των γεωδαιτικών που διέρχονται από ένα συγκεκριμένο κόμβο n_i γράφουμε $g_{jk}(n_i)$. Έτσι έχουμε τον τύπο για αυτή τη κεντρικότητα που προτάθηκε από τον Freeman:

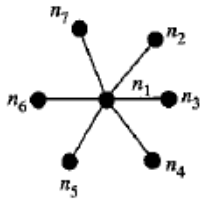
$$C_B(n_i) = \sum_{j<k} \frac{g_{jk}(n_i)}{g_{jk}},$$

δηλαδή είναι άθροισμα των γεωδαιτικών που διέρχονται από τον κόμβο n_i προς τις συνολικές γεωδαιτικές για κάθε ζευγάρι κόμβων.
Η κανονικοποιημένη μορφή του παραπάνω τύπου είναι:

$C'_B = \frac{C_B(n_i)}{\frac{(g-1)(g-2)}{2}}$, με τιμές ανάμεσα στο 0 και στο 1 για μη-κατευθυνόμενα γραφήματα και

$C'_B = \frac{C_B(n_i)}{(g-1)(g-2)}$ για κατευθυνόμενα γραφήματα.

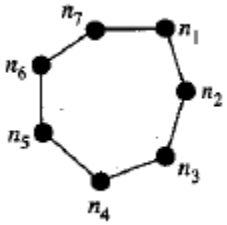
Σε σύγκριση με την παραπλήσια κεντρικότητα στον παραπάνω τύπο δεν υπάρχει πρόβλημα με τους απομονωμένους κόμβους.
Στο παρακάτω γράφημα(αστέρι) έχουμε:



Betweenness Centrality

n_i	C'_B
1	1.0
2	0
3	0
4	0
5	0
6	0
7	0

Ενώ στο γράφημα(κύκλο) έχουμε:



Betweennes Centrality

n_i	C'_B
1	0.2
2	0.2
3	0.2
4	0.2
5	0.2
6	0.2
7	0.2

Για την περίπτωση που έχουμε βάρη στο γράφημα, έχουν προτείνει, όπως προηγουμένως, ένα εναλλακτικό υπολογισμό οι T.Opsahl ,F.Agneessens, J.Skvoretz , όπου ορίζεται ως εξής:

$$C_B^{wa}(n_i) = \frac{g_{jk}^{wa}(n_i)}{g_{jk}^{wa}},$$

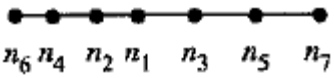
με το $g_{jk}^{wa}(n_i)$ να είναι ο αριθμός των γεωδαιτικών που διέρχονται από τον κόμβο n_i και το g_{jk}^{wa}ο αριθμός των συνολικών γεωδαιτικών, ενώ η παράμετρος *α* χρησιμοποιείται με τον ίδιο τρόπο όπως προηγουμένως.

Γράφημα

Όπως και στις δύο προηγούμενες περιπτώσεις ο ορισμός της between-nesscentrality μπορεί να γενικευτεί και για γράφημα με σκοπό την σύγκρισή τους. Ο τύπος που πρότεινε ο Freeman είναι ο εξής:

$$C_B = \frac{2\sum_{i=1}^{g}[C_B(n^*) - C_B(n_i)]}{[(g-1)^2(g-2)]}.$$

Για το γράφημα ευθεία έχουμε $C_B = 0.311$.



Κεντρικότητα πληροφορίας (Information Centrality)

Κόμβου

Αυτό το μέτρο προτάθηκε από τους Stephenson και Zelen, για τον υπολογισμό του χρειάζεται να ορίσουμε ένα νέο πίνακα με τα παρακάτω στοιχεία:

$a_{ii} = 1 +$το άθροισμα των τιμών τον ακμών που προσπίπτουν στον κόμβο n_i και

$$\alpha_{ij} = \begin{cases} 1 \; \alpha\nu \; o\iota \; n_i \; \kappa\alpha\iota \; n_j \; \delta\varepsilon\nu \; \sigma\upsilon\nu\delta\acute{\varepsilon}o\nu\tau\alpha\iota \\ 1 - x_{ij} \alpha\nu \; o\iota \; n_i \; \kappa\alpha\iota \; n_j \; \sigma\upsilon\nu\delta\acute{\varepsilon}o\nu\tau\alpha\iota \end{cases},$$

όπου x_{ij} είναι η τιμή της ακμής.

Με τον τρόπο αυτό δημιουργείται ένας νέος πίνακας A και ορίζουμε ως C τον αντίστροφο του (πρέπει να προσέξουμε να μην υπάρχουν γραμμές στον A μόνο με μηδενικά γιατί τότε δεν αντιστρέφεται, άρα στον παραπάνω πίνακα θα πρέπει να εξαιρέσουμε τους απομονωμένους κόμβους). Επιπλέον ορίζουμε με βάση τα στοιχεία του C: $T = \sum_{i=1}^{g} c_{ii}$ και $R = \sum_{j=1}^{g} c_{ij}$ με το T να είναι το άθροισμα των διαγώνιων στοιχείων και το R να είναι το άθροισμα των στοιχείων κάθε σειράς(είναι ίσο για κάθε σειρά). Έτσι προκύπτει τελικά:

$C_I(n_i) = \frac{1}{c_{ii} + \frac{T-2R}{g}}$,ως η κεντρικότητα πληροφορίας για τον κόμβο i.

Ενώ η κανονικοποιημένη κεντρικότητα πληροφορίας δίνεται από τον τύπο:

$C_I'(n_i) = \frac{C_I(n_i)}{\sum_i C_I(n_i)}$, με τιμές ανάμεσα στο 0 και στο 1.

Για το γράφημα αστέρι που είχαμε παραπάνω προκύπτουν οι εξής τιμές:

Κεντρικότητες Πληροφορίας για το Γράφημα Αστέρι

n_i	$C_I'(n_i)$
1	0.2340
2	0.1277
3	0.1277
4	0.1277
5	0.1277
6	0.1277
7	0.1277

Γραφήματος

Η κεντρικότητα πληροφορίας για γράφημα προσδιορίστηκε από του Stephenson και Zelen ως ο μέσος όπως δίνεται από :

$\overline{C_I} = \sum_i C_I(n_i)$.

Επίσης μπορεί να δοθεί από τη διασπορά:

$S_I^2 = \frac{\sum_{i=1}^{g} (C_I'(n_i) - \overline{C_I})^2}{g}$.

Για τα γραφήματα του αστεριού, του κύκλου και της ευθείας έχουμε διασπορές 0.001614,0.000986 και 0.0 αντίστοιχα. Συνεπώς το αστέρι είναι το πιο ετερογενές και ο κύκλος το λιγότερο.

Κεντρικότητα Ιδιοδιανυσμάτων (Eigenvector Centrality)

Έστω x_i ο βαθμός κεντρικότητας ιδιοδιανυσμάτων του κόμβου n_i. Ορίζουμε ως πίνακα A, τον πίνακα με στοιχεία A_{ij} = το βάρος της σύνδεσης του κόμβου n_i με τον n_j, αν δεν συνδέονται οι κόμβοι αυτοί έχουμε $A_{ij} = 0$.
Έτσι για τον κόμβο n_i ορίζεται η κεντρικότητα ιδιοδιανυσμάτων:

$x_i = \frac{1}{\lambda}\sum_{j\in M(i)} x_j = \frac{1}{\lambda}\sum_{j=1}^{N} A_{ij}x_j$,

όπου M(i) είναι το σύνολο των κόμβων που είναι συνδεδεμένος ο n_i και το λ είναι μία σταθερά. Τις παραπάνω εξισώσεις μπορούμε να τις γράψουμε και σε διανυσματική μορφή ως εξής:

$\boldsymbol{x} = (x_1.x_2 \ldots., x_g)$, $\boldsymbol{x} = \frac{1}{\lambda}A\boldsymbol{x}$ή σε εξίσωση ιδιοδιανυσμάτων $A\boldsymbol{x} = \lambda\boldsymbol{x}$.

Από την παραπάνω εξίσωση θα προκύψουν αρκετές ιδιοτιμές με λύσεις τα ιδιοδιανύσματα τους, όμως με βάση το θεώρημα Perron-Frobenius για να έχουμε θετικές κεντρικότητες πρέπει να επιλέξουμε τη μεγαλύτερη ιδιοτιμή λ, τότε το ιδιοδιάνυσμα που θα μας δώσει θα αποτελεί και τις κεντρικότητες ιδιοδιανυσμάτων.
Δηλαδή κάθε συνιστώσα i του ιδιοδιανύσματος **x** θα αποτελεί την κεντρικότητα ιδιοδιανυσμάτων του αντίστοιχου κόμβου.
Για το Γράφημα 2 έχουμε ιδιοτιμές { -10.24 , -7.38 , -3.71 , 0.88 , 5.47 , 14.93 }, επιλέγουμε την μεγαλύτερη, δηλαδή λ=14.93 και από το ιδιοδιάνυσμα που προκύπτει έχουμε τις παρακάτω κεντρικότητες :

Κεντρικότητες Ιδιοδιανυσμάτων

n_i	x_i
1	0.83
2	1.07
3	1.47
4	0.90
5	1.21
6	1

Η κεντρικότητα ιδιοδιανυσμάτων μας δίνει ένα βελτιωμένο υπολογισμό του βαθμού κεντρικότητας, γιατί λαμβάνει υπόψη και την κεντρικότητα των γειτόνων ενός κόμβου.

Μέτρα υπολογισμού Πρεστίζ

Βαθμός πρεστίζ (Degree Prestige)

Το πιο απλό μέτρο του πρεστίζ είναι το άθροισμα των ακμών που κατευθύνονται σε ένα κόμβο. Όταν ένας κόμβος έχει υψηλό βαθμό πρεστίζ σημαίνει ότι πολλοί κόμβοι(άτομα) επικοινωνούν με αυτόν, άρα έχει μεγάλη επιρροή μέσα στο δίκτυο. Έτσι ορίζουμε :

$P_D(n_i) = \sum_{j=1}^{g} x_{ji}.$

Και η κανονικοποιημένη μορφή είναι :

$P'_D(n_i) = \frac{\sum_{j=1}^{g} x_{ji}}{g-1}$ με τιμές ανάμεσα στο 0 και στο 1.

Για το Γράφημα 1 έχουμε:

Βαθμοί Πρεστίζ Κόμβων

n_i	$P_D(n_i)$	$P'_D(n_i)$
A	1	0.2
B	2	0.4
C	2	0.4
D	2	0.4
E	2	0.4
F	2	0.4

Εγγύτητα πρεστίζ (Proximity Prestige)

Κόμβος

Για το μέτρο αυτό θα ορίσουμε αρχικά τον τομέα επιρροής ενός κόμβου I_i. ΤοI_i ορίζεται ως ο αριθμός των κόμβων που κατευθύνονται στον κόμβο n_i, δηλαδή υπάρχουν μονοπάτια που κατευθύνονται σε αυτόν. Η εγγύτητα πρεστίζ ορίζει το πόσο κοντά είναι οι κόμβοι του δικτύου στο κόμβο που μελετάμε και ο τύπος της είναι ο εξής :

$P_P(n_i) = \frac{\frac{I_i}{g-1}}{\frac{\sum_{j=1}^{g} d(n_j,n_i)}{I_i}}$ με τιμές ανάμεσα στο 0 και στο 1.

Αν ένας κόμβος είναι απρόσιτος τότε η εγγύτητα πρεστίζ του είναι 0, ενώ όλοι οι κόμβοι ενός δικτύου επικοινωνούν με κάποιο κόμβο τότε αυτός έχει εγγύτητα πρεστίζ 1.

Γραφήματος

Για ένα γενικό μέτρο εγγύτητας πρεστίζ για το γράφημα μπορούμε να χρησιμοποιήσουμε τη διασπορά της εγγύτητας πρεστίζ των κόμβων. Έτσι έχουμε :

$\overline{P_P} = \sum_{i=1}^{g} \frac{P_p(n_i)}{g}$ και $S_p^2 = \frac{\sum_{i=1}^{g} (P_p(n_i) - \overline{P_P})^2}{g}$ με τιμές ανάμεσα στο 0 και στο 1.

Η διασπορά αυτή μετράει την ετερογένεια του γραφήματος με βάση την εγγύτητα του πρεστίζ.

Τάξη ή στάτους πρεστίζ (Status or Rank Prestige)

Η μέθοδος αυτή μας δίνει το πρεστίζ ενός κόμβου βασιζόμενη στο πρεστίζ των κόμβων που κατευθύνονται σε αυτόν, δηλαδή στη μέθοδο αυτή δεν μας ενδιαφέρει μόνο ο ίδιος ο κόμβος αλλά τι γίνεται γύρω του. Η τάξη ορίζεται ως ο γραμμικός συνδυασμός των ακμών που κατευθύνονται σε ένα κόμβο:

$$P_R(n_i) = x_{1i}P_R(n_1) + x_{2i}P_R(n_2) + x_{3i}P_R(n_3) + \cdots + x_{gi}P_R(n_g)$$

Η τάξη ενός κόμβου είναι συνάρτηση των τάξεων των κόμβων που επικοινωνούν με αυτόν. Έτσι δημιουργείται μία εξίσωση με g αγνώστους και g εξισώσεις.

$$P = \left(P_R(n_1), P_R(n_2), \dots, P_R(n_g)\right)^T$$
$$P = X^T P$$

Αυτή η εξίσωση είναι η χαρακτηριστική εξίσωση που χρησιμοποιούμε για να βρούμε ιδιοτιμές και ιδιοδιανύσματα του πίνακα X^T. Από τις ιδιότιμές αυτές, επιλέγουμε την μεγαλύτερη και το ιδιοδιάνυσμα που μας δίνει αποτελείται από τις τιμές τις τάξης πρεστίζ, δηλαδή η i συνιστώσα του ιδιοδιανύσματος είναι η τάξη πρεστίζ του i κόμβου.
Για το Γράφημα 1 έχουμε τη μεγαλύτερη ιδιοτιμή λ=1.62 και το ιδιοδιάνυσμα μας δίνει τις τιμές της τάξης του πρεστίζ για τους κόμβους.

Τάξη Πρεστίζ Κόμβων

n_i	x_i
A	0
B	0
C	0
D	0.62
E	1
F	1

Συντελεστής Ομαδοποίησης (Local clustering Coefficient)

Ο συντελεστής ομαδοποίησης ενός κόμβου μας δείχνει κατά πόσο οι «γείτονες» του κόμβου είναι δυνατόν να γίνουν κλίκα, δηλαδή να αποτελούν μια μικρή ομάδα που επικοινωνούν όλοι μεταξύ τους. Η γειτονιά ενός κόμβου n_i ορίζεται ως οι κόμβοι που είναι άμεσα συνδεδεμένοι με αυτόν :

$N_i = \{n_j : e_{ij} \in E \ \wedge e_{ji} \in E\}$(όπου Ε είναι το σύνολο των ακμών ενός γραφήματος)
$k_i = |N_i|$, ο αριθμός των ακμών των γειτόνων.

Έτσι ο συντελεστής ομαδοποίησης για ένα κόμβο n_i είναι :

$$C_i = \frac{|\{e_{jk}\}|}{k_i(k_i - 1)} : u_j, u_k \in N_i, e_{jk} \in E$$

Για το Γράφημα 1 έχουμε :

n_i	N_i	C_i
A	{C}	1
B	{C}	1
C	{A}	1
D	{F}	1
E	{F}	1
F	{D}	1

Βαθμοί Πρεστίζ Κόμβων

Το αποτέλεσμα αυτό ($C_i = 1 \ \forall \ i$) προκύπτει από το γεγονός ότι στο γράφημα αυτό υπάρχουν δύο κλίκες , η {A,B,C} και {D,E,F}.

Σχεδίαση Γραφημάτων

Force-Directed Algorithms

Το δίκτυο που προκύπτει από τα δεδομένα μας αποτελεί ένα κατευθυνόμενο γράφο και για την απεικόνιση αυτού χρησιμοποιώ ένα αλγόριθμο που ανήκει στην κλάση των force-directed αλγορίθμων. Οι αλγόριθμοι αυτοί αποσκοπούν στην σχεδίαση ενός γράφου με ένα αισθητικά αποδεκτό τρόπο. Ο σκοπός τους είναι να τοποθετήσουν τους κόμβους του γράφου σε δισδιάστατο ή τρισδιάστατο χώρο έτσι ώστε όλες οι ακμές να έχουν περίπου το ίδιο μήκος και να υπάρχουν όσο το δυνατόν λιγότερες διατμήσεις των ακμών. Ο πρώτος αλγόριθμος της κλάσης αυτής προτάθηκε το 1980 από τον Eades και τον Kamada-Kawai με την ονομασία spring-embedder και στην συνέχεια το 1990 δόθηκε ο όρος force-directed από του Fruchterman&Reingold.(Fruchterman, etal., 1991)

Η λειτουργία τον force-directed αλγορίθμων βασίζεται στην προσομοίωση ενός φυσικού σύστημα δυνάμεων ανάμεσα στους κόμβους και στις ακμές. Την πιο συνηθισμένη μέθοδος αποτελεί η αναπαράσταση των ακμών ως ελατήρια που ασκούν τις αντίστοιχες δυνάμεις ανάμεσα στου κόμβους, δηλαδή το νόμο του Hooke, και οι κόμβοι θεωρούνται ηλεκτρικά φορτισμένα σωματίδια όπου απωθούνται με βάση τον νόμο του Coulomb. Στην συνέχεια ολόκληρος ο γράφος προσομοιώνεται ως το σύστημα που περιγράφηκε και με βάση τις παραπάνω δυνάμεις μετακινούνται στις κατάλληλες θέσεις οι κόμβοι, έλκοντας ή απωθώντας ο ένας τον άλλο. Η διαδικασία αυτή επαναλαμβάνεται έως ότου επιτευχθεί ισορροπία και οι σχετικές θέσεις τον κόμβων δεν αλλάζουν ανάμεσα στις επαναλήψεις.

Ένα εναλλακτικό μοντέλο εισάγει στις δυνάμεις του ελατηρίου την έννοια του ιδανικού φυσικού μήκους ελατηρίου, δηλαδή την μικρότερη απόσταση όπου πρέπει να βρεθούν οι κόμβοι για να επιτευχθεί η ισορροπία. Η απόσταση αυτή μπορεί να καθοριστεί με βάση τα δεδομένα και την σχέση που υπάρχει ανάμεσα στους δύο κόμβους, στην περίπτωση των δεδομένων που επεξεργάζομαι η απόσταση αυτή αναλογεί στην επικοινωνία(σε δευτερόλεπτα) μεταξύ των κόμβων.

Ένας force-directed γράφος μπορεί να περιέχει και άλλες δυνάμεις πέρα από τις καθιερωμένες του ελατηρίου και της ηλεκτρικής απώθησης. Μερικά παραδείγματα αποτελούν τα λογαριθμικά ελατήρια, οι βαρυτικές δυνάμεις, τα μαγνητικά πεδία και τα ηλεκτρικά φορτισμένα ελατήρια. (Di Battista, et al., 1999)

Πλεονεκτήματα Force-Directed Αλγορίθμων

- Ποιοτικά αποτελέσματα για γράφους μεσαίου μεγέθους, δηλαδή αποτελούμενους από 50-100 κόμβους.
- Ευελιξία. Οι Force-Directed αλγόριθμοι μπορούν εύκολα να προσαρμοστούν για να εκπληρώσουν επιπλέον αισθητικά κριτήρια με την απλή προσθήκη περαιτέρω δυνάμεων ή την προσαρμογή των υπαρχόντων.
- Κατανόηση. Λόγω του φυσικού υποβάθρου των αλγορίθμων αυτών η κατανόηση του τρόπου λειτουργίας τους είναι απλή.
- Ευκολία. Οι αλγόριθμοι αυτοί είναι απλοί και μπορούν να εφαρμοστούν μέσα σε λίγες γραμμές κώδικα .
- Διαδραστικότητα. Μέσω της σχεδίασης των ενδιάμεσων φάσεων που βρίσκεται το σύστημα μπορεί να παρουσιαστεί η κίνηση των κόμβων και να κατανοηθεί ευκολότερα η λειτουργία τους. Επιπλέον υπάρχει η δυνατότητα της αλλαγής της θέσης ενός κόμβου με συνέπεια το σύστημα να αναπροσαρμοστεί και να κινηθεί ξανά έως ότου βρεθεί σε ισορροπία.

Μειονεκτήματα Force-Directed Αλγορίθμων

- Υπολογιστικός Χρόνος. Ένας τυπικός force-directed αλγόριθμος έχει πολυπλοκότητα της τάξης $O(n^3)$, όπου n αποτελεί τον αριθμό των κόμβων του γραφήματος. Η πολυπλοκότητα αυτή προκύπτει γιατί ο αριθμός των επαναλήψεων εκτιμάται O(n) και σε επανάληψη υπολογίζονται οι δυνάμεις για κάθε ζευγάρι κόμβων.
- Περιορισμός στο μέγεθος του γράφου. Από την Βιβλιογραφία προκύπτει πως οι αλγόριθμοι της κλάσης αυτής μπορούν να απεικονίσουν σωστά γράφους μεγέθους έως 100 κόμβων. (Kobourov, 2012)

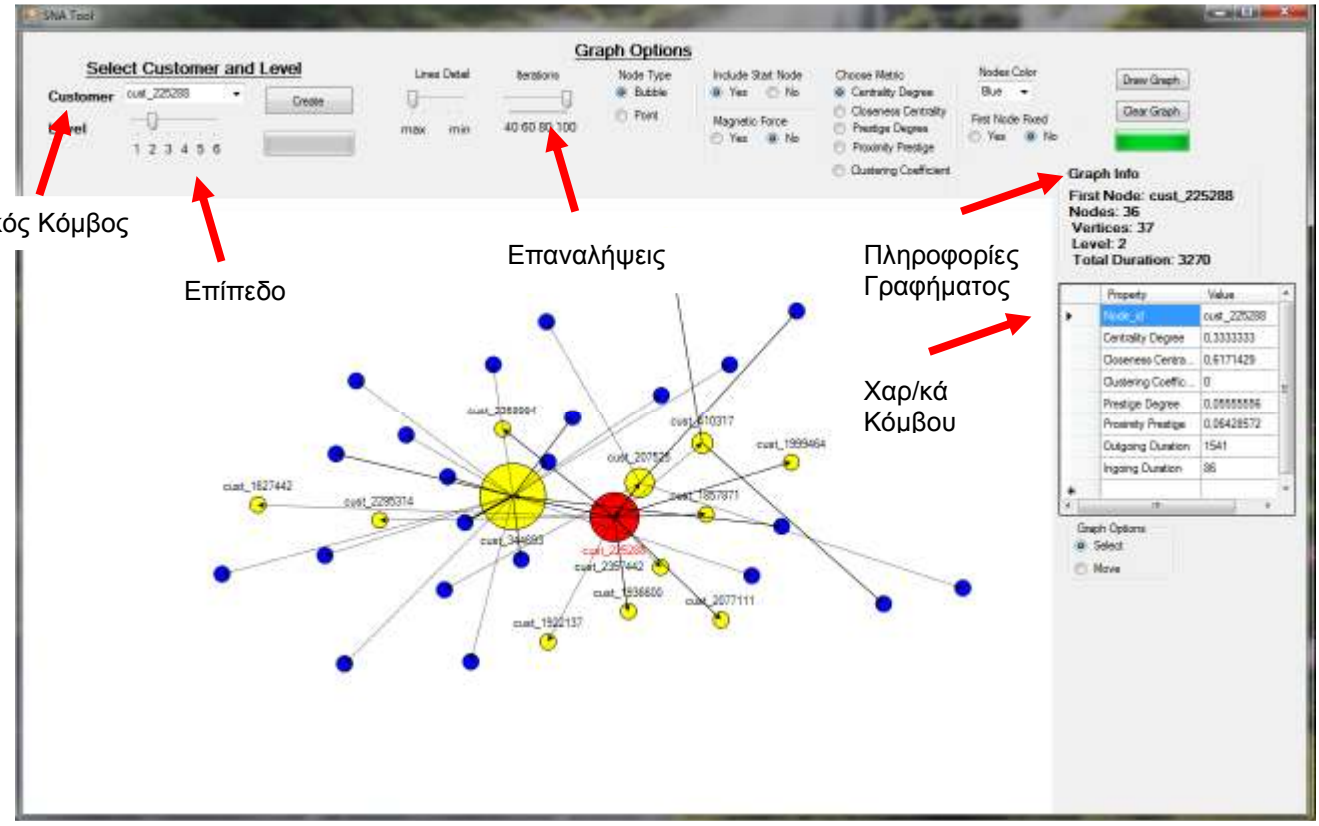
Αλγόριθμος

1. Τοποθέτηση των κόμβων σε τυχαίες θέσεις
2. Επανάληψη M φορές
3. Για κάθε κόμβο n_1 από το σύνολο των κόμβων
4. Για κάθε κόμβο n_2 από το σύνολο των κόμβων
5. Αν $n_1!=n_2$
6. Αν n_1 και n_2 συνδέονται
7. Υπολόγισε την F_{atr} για το ζευγάρι αυτό
8. Αλλιώς
9. Υπολόγισε την F_{rep} για το ζευγάρι αυτό
10. Επόμενος κόμβος
11. Μετακίνηση του n_1 κατά a προς την κατεύθυνση της δύναμης
12. Επόμενος κόμβος
13. Επόμενη επανάληψη
14. Τοποθέτηση των κόμβων στην τελική τους θέση
15. Απεικόνιση των κόμβων και των ακμών με κατάλληλο εργαλείο

Στους αλγορίθμους αυτού του τύπου οι δυνάμεις μπορούν να διαφέρουν ανάλογα με τα αισθητικά αποτελέσματα που θέλουμε να επιτευχθούν, για παράδειγμα η δύναμη του ελατηρίου μπορεί να είναι ο κλασσικός νόμος του Hookeή κάποια αντίστοιχης συμπεριφοράς όπως ο φυσικός λογάριθμος. Επιπλέον υπάρχει η δυνατότητα προσθήκης περαιτέρω δυνάμεων για ορισμένα εξειδικευμένα αποτελέσματα, αυτές οι δυνάμεις μπορεί να είναι μαγνητικά ή ηλεκτρικά πεδία.

Περιγραφή Εφαρμογής

Για την Ανάλυση Κοινωνικών Δικτύων χρειάζονται κατάλληλα εργαλεία για την εξαγωγή συμπερασμάτων πάνω σε ένα Δίκτυο, όπως η απεικόνιση του δικτύου και ο υπολογισμός τον μέτρων κεντρικότητας και πρεστίζ. Η εφαρμογή που ανέπτυξα περιλαμβάνει ορισμένες λειτουργίες προς την κατεύθυνση αυτή. Η λειτουργία της βασίζεται στη δημιουργία ενός δίκτυο γύρω από ένα αρχικό κόμβο μέχρι ένα προκαθορισμένο επίπεδο και στη συνέχεια η απεικόνιση του δικτύου αυτού ανάλογα με τις κατάλληλες ρυθμίσεις. Η λειτουργία της εφαρμογής χωρίζεται σε τρία βασικά μέρη, την σύνδεση με την βάση δεδομένων και την ανάκτηση του κατάλληλου δικτύου, τον υπολογισμό των μέτρων της Ανάλυσης Κοινωνικών Δικτύων και της απεικόνισης του Δικτύου αυτού σύμφωνα με τις ανάλογες απαιτήσεις. Το GUI της εφαρμογής παρουσιάζεται παρακάτω με ένα screenshot, ενώ η περιγραφή της λειτουργίας της ακολουθεί στη συνέχεια.



Ανάκτηση Δεδομένων

Αρχικά παρουσιάζονται τα δεδομένα που χρησιμοποιώ για την εφαρμογή αυτή. Τα δεδομένα αυτά μου δόθηκαν από την εταιρεία Datamine όπου έκανα πρακτική, είναι κωδικοποιημένα και έχουν την παρακάτω μορφή πίνακα από το SQLServer :

	anum	bnum	call_type	call_dur
1	cust_225288	cust_2077111	2	16
2	cust_221590	cust_2130052	9	0
3	cust_295494	cust_1131052	2	29
4	cust_406266	cust_1657493	2	2
5	cust_408596	cust_2356581	2	0
6	cust_366683	cust_2637736	2	63
7	cust_57894	cust_562198	2	0
8	cust_63613	cust_389831	2	0
9	cust_437000	cust_2084964	2	0
10	cust_273168	cust_1394990	2	3

Όπου τα *anum*, *bnum* αποτελούν τους κόμβους του δικτύου, το *call_type* το είδος της συνδιάλεξης (το 2 αποτελεί τις κλήσεις και το 9 τα μηνύματα) και τέλος το *call_dur* την διάρκεια σε δευτερόλεπτα της συνδιάλεξης.

Για την μεταφορά των παραπάνω δεδομένων και την «μετάφραση» τους σε ένα αντικείμενο της C# δημιούργησα μία νέα κλάση την κλάση Entities όπου ορίζεται όσα αντικείμενα χρειάζονται για τη δημιουργία ενός γράφου και της απεικόνισης αυτού.

Αρχικά το αντικείμενο *Graph* αποτελεί το γράφο και περιέχει μια λίστα με όλους τους κόμβους ,αντικείμενο *Node*, και ένα int με το συνολικό call. Το αντικείμενο *Node* έχει τις παρακάτω ιδιότητες:

Id: Είναι τύπου string και αποτελεί το όνομα του κόμβου όπως στη βάση δεδομένων

Out_Adjacency: Είναι μία λίστα από Adjacency, δηλαδή από τους γείτονες προς τους οποίους επικοινωνεί ο κόμβος.

In_Adjacency: Είναι μία λίστα από Adjacency όσων κόμβων επικοινωνούν με τον κόμβο αυτό.

Level: Είναι ένα byte και αποτελεί το level που ανήκει ο κόμβος με βάση τον αρχικό.

Το αντικείμενο *Adjacency* που ανέφερα παραπάνω έχει τις εξής δύο ιδιότητες :

Id: Είναι τύπου string με το όνομα του κόμβου

Dur: Είναι τύπου int και αποτελεί το σύνολο της επικοινωνίας των δύο κόμβων σε δευτερόλεπτα.

Για την σύνδεση με τη βάση δεδομένων χρησιμοποιώ την κλάση DBManager. Στην κλάση αυτή ανάλογα με το level που επιλέγουμε στην εφαρμογή δημιουργείται δυναμικά ένα string που αποτελεί το query για την βάση δεδομένων. Ένα παράδειγμα του query αυτού για level 3 είναι το ακόλουθο:

select anum,bnum,call_dur from final where anum in ('cust_185866')
union select anum,bnum,call_dur from final where anum in (select bnum from final where anum in ('cust_185866'))
union select anum,bnum,call_dur from final where anum in (select bnum from final where anum in (select bnum from final where anum in ('cust_185866')))

Στη συνέχεια τα αποτελέσματα του παραπάνω query διαβάζονται ανά σειρά και καταχωρούνται ως νέα *Node* ή προστίθενται νέα δεδομένα στην υπάρχουσα λίστα του *Graph*. Με τον τρόπο αυτό στο τέλος της διαδικασίας αυτής υπάρχει αποθηκευμένο το δίκτυο όλο σαν αντικείμενο στην C# και ακολουθεί το υπολογιστικό μέρος πάνω στο δίκτυο.

Υπολογισμός Μέτρων

Στο μέρος αυτό υπολογίζονται ορισμένα από τα μέτρα της Ανάλυσης Κοινωνικών Δικτύων, όπως ο Βαθμός Κεντρικότητας(Centrality Degree), η Παραπλήσια Κεντρικότητα (Closeness Centrality), ο Βαθμός Πρεστίζ (Prestige Degree), η Εγγύτητα Πρεστίζ (Proximity Prestige) και ο Συντελεστής Ομαδοποίησης (Clustering Coefficient). Για την αποθήκευση των μέτρων αυτών έφτιαξα στην κλάση Entities ένα νέο αντικείμενο, το *Degree*, που έχει ιδιότητες τις εξής:

Id: Είναι τύπου string και αποτελεί την ταυτότητα του κόμβου
Deg: Είναι τύπου float και αποτελεί το αντίστοιχο μέτρο για τον κόμβο.

Centrality Degree

Για το Βαθμό Κεντρικότητας(Centrality Degree) ο υπολογισμός είναι αρκετά εύκολος γιατί τα αναγκαία στοιχεία για το μέτρο αυτό υπάρχουν ήδη από την δημιουργία του κόμβου. Δηλαδή έχουμε από το *Out_Adjacency* τον αριθμό των κόμβων με τους οποίους επικοινωνεί και στο αντικείμενο Graph έχουμε το σύνολο των κόμβων, άρα αρκεί μία απλή διαίρεση για το αποτέλεσμα.

Ψευδοκώδικας
1. *Για κάθε κόμβο n_1 του γράφου*
2. *Ορισμός ως deg το άθροισμα των out_Adjacency*
3. *Αν deg!=0*
4. *CentralityDegree= deg/n, όπου n=συνολικός αριθμός κόμβων*
5. *Επόμενος κόμβος*

Closeness Centrality

Το επόμενο μέτρο αποτελεί η Παραπλήσια Κεντρικότητα (Proximity Prestige), για την οποία χρειάζεται να έχουμε τις γεωδαιτικές ανάμεσα σε κάθε ζευγάρι κόμβων. Έτσι αρχικά υπολογίζονται οι γεωδαιτικές με μία μέθοδο GetGeod που επιστρέφει ένα Dictionary<string,int> και για το δεδομένο κόμβο μας δίνει ως Key του Dictionary το κόμβο με τον οποίο συνδέεται και ως Value την απόσταση τους.

Ψευδοκώδικας

```
1.  Για κάθε κόμβο n1 του γράφου
2.      Τρέξετην GetGeod=geod_dictionary
3.          Αν geod_dictionary!=null
4.              int d=0
5.                  Για κάθε ζευγάρι του geod_dictionayr
6.                      d+=geod_dictionary.Value
7.                  Επόμενο ζευγάρι
8.              deg = (geod.dictionary.Keys.Count^2)/(n-1)*d
9.  Επόμενος κόμβος
```

Clustering Coefficient

Στη συνέχεια υπολογίζεται ένα ενδιάμεσο μέτρο που δεν ανήκει στην κατηγορία των Centrality ή στα Prestige και είναι το Clustering Coefficient. Για το Clustering Coefficient απαιτείται να βρεθεί η γειτονιά του κόμβου, δηλαδή με όσους κόμβους υπάρχει αμφίδρομη επικοινωνία και πόσοι από τους κόμβους της γειτονιάς αυτής συνδέονται μεταξύ τους.

Ψευδοκώδικας

```
1.  Για κάθε κόμβο n1 του γράφου
2.      List<string> Neighboorsμε id τωνγειτόνων,int e=0,int k=0.
3.          Για κάθεαAdjacecny από το Out_Adjacency του n1
4.              Αν n1∈In_Adjacency του a
5.                  k+=1
6.                  Neighboors.Add(a)
7.          Επόμενο Adjacency
8.          Αν k>1
9.            Για κάθε neigh του Neighboors
10.               Για κάθε nei του Neighboors
11.                   Αν nei∈Out_Adjacency του neigh
12.                       e+=1
13.               Επόμενο string
14.           Επόμενο string
15.         Αν e!=0
16.             deg = (e)/k*(k-1)
17. Επόμενος κόμβος
```

Prestige Degree

Το πρώτο μέτρο Prestige αποτελεί ο Βαθμός Πρεστίζ και είναι αρκετά εύκολος ο υπολογισμός του όπως και στο Centrality Degree.

Ψευδοκώδικας

```
6.  Για κάθε κόμβο n1 του γράφου
7.      Ορισμός ως deg το άθροισμα των in_Adjacency
8.          Αν deg!=0
9.              CentralityDegree= deg/n, όπου n=συνολικός αριθμός κόμβων
10. Επόμενος κόμβος
```

Proximity Prestige

Ο υπολογισμός είναι ο ίδιος από την πλευρά των In_Adjacency όμως αυτή την φορά. Έτσι υπολογίζεται αρχικά ο τομέας επιρροής του κόμβου, δηλαδή πόσοι κόμβοι επικοινωνούν με τον κόμβο. Στη συνέχεια υπολογίζονται οι γεωδαιτικές ανάμεσα στον κόμβο και τον τομέα επιρροής του.

Ψευδοκώδικας

1. *Για κάθε κόμβο n_1 του γράφου*
2. *Τρέξετην GetGeod=geod_dictionary*
3. *Αν geod_dictionary!=null*
4. *int d=0*
5. *Για κάθε ζευγάρι του geod_dictionayr*
6. *d+=geod_dictionary.Value*
7. *Επόμενο ζευγάρι*
8. *deg = (geod.dictionary.Keys.Count^2)/(n-1)*d*
9. *Επόμενος κόμβος*

Η διαφορά με το Ψευδοκώδικα της Παραπλήσιας Κεντρικότητας έγκειται στη μέθοδο GetGeod όπου για την συγκεκριμένη περίπτωση χρησιμοποιεί τα In_Adjacency.

Απεικόνιση Γραφήματος

Στο Τρίτο μέρος της εφαρμογής πραγματοποιείται η απεικόνιση του δικτύου αυτού που έχει επιλεχθεί προηγουμένως. Η απεικόνιση αυτή υπόκειται σε αρκετές διαφοροποιήσεις ανάλογα τις επιλογές που μπορούμε να κάνουμε στο GUI της εφαρμογής. Οι επιλογές αυτές βασίζονται στις διαφορετικές απαιτήσεις που μπορεί να έχει ο χρήστης για την τελική απεικόνιση και ανάλογα με την ανάλυση που θέλει να γίνει παρουσιάζεται διαφορετική εικόνα. Εκτός των ρυθμίσεων που υπάρχουν για το δίκτυο και την ανάλυση του, στο GUI της εφαρμογής υπάρχουν ρυθμίσεις και για τα αισθητικά στοιχεία του δικτύου όπως το πάχος των ακμών ,ο τύπος και το χρώμα των κόμβων.

Για την απεικόνιση του γράφου χρησιμοποιείται ένας Force-Directed αλγόριθμος και οι δυνάμεις που χρησιμοποιούνται σε αυτόν είναι οι παρακάτω:

$$F_{atr}(d_{ij}) = \ln\left(\frac{d_{ij}}{l_{ij}}\right) \qquad F_{rep}(d_{ij}) = c * \frac{1}{d_{ij}^2}$$

Η δύναμη F_{atr} ασκείται ανάμεσα στους κόμβους που επικοινωνούν και ως l_{ij} ορίζεται το φυσικό μήκος του ελατηρίου, δηλαδή η απόσταση η οποία είναι ιδανικό να έχουν οι δύο κόμβοι. Αυτή η λογαριθμική δύναμη προτιμάται από τον κλασσικό νόμο του Hooke γιατί όταν η απόσταση των κόμβων είναι μεγάλη ο κλασσικό νόμος του Hooke αποτελεί πολύ ισχυρή δύναμη και επιφέρει μεγάλες αλλαγές στο γράφο. Ενώ για τις απωθητικές δυνάμεις, τη δύναμη F_{rep} έχουμε μία δύναμη αντιστρόφου της απόστασης με σκοπό να αυξάνεται όσο οι αποστάσει μικραίνουν και οι κόμβοι τείνουν να φτάσουν ο ένας τον άλλο. Οι απωθητικές δυνάμεις ασκούνται μεταξύ στους κόμβους που δεν έχουν άμεση επικοινωνία για να αποφευχθεί το φαινόμενο που μόλις περιγράφθηκε.

Οι σταθερές που υπάρχουν τόσο στον ορισμό του μήκους του ελατηρίου όσο και στις απωθητικές δυνάμεις χρειάστηκαν πολλές δοκιμές για να προσδιοριστούν και να έχουμε το επιθυμητό αποτέλεσμα. Ακόμα και στην τελική έκδοση της εφαρμογής παρουσιάζεται

μερικές φορές το φαινόμενο μερικοί κόμβοι να αλληλεπικαλύπτονται, το γεγονός αυτό μπορεί να αντιμετωπιστεί είτε με την αύξηση του φυσικού μήκους του ελατηρίου αν οι δύο κόμβοι συνδέονται ή με την αύξηση της σταθεράς στις απωθητικές δυνάμεις.

Η απεικόνιση γίνεται ως εξής και η μέθοδος αυτή ονομάζεται «Follow Your Nose». Αρχικά οι κόμβοι τοποθετούνται σε τυχαίες θέσεις μέσα στο γράφο, οι οποίες απεικονίζονται στο παράθυρο, και στη συνέχεια υπολογίζονται οι παραπάνω δυνάμεις ανά επανάληψη, με βάση τις δυνάμεις αυτές ωθούνται προς τις νέες θέσεις τους έως ότου φτάσουν σε ένα επίπεδο ισορροπίας. Ανά 5 επαναλήψεις εμφανίζονται στο παράθυρο οι νέες θέσεις των κόμβων έτσι ώστε να φαίνεται η «κίνηση» προς την ιδανική θέση. Ο αριθμός των επαναλήψεων ορίζεται από τον χρήστη μέσω του GUI της εφαρμογής και αποτελεί ένα μέτρο της ταχύτητας ή της ποιότητας της τελικής απεικόνισης. Για την ομοιομορφία της μεταβολής ανάμεσα στις επαναλήψεις, δηλαδή τη μη ύπαρξη μεγάλων «αλμάτων» σε κάθε επανάληψη ασκείται μόνο ένα ποσοστό της δύναμης σε κάθε επανάληψη.

Οι επιπλέον ρυθμίσεις που υπάρχουν στην εφαρμογή αφορούν ποιο μέτρο θα χρησιμοποιήσουμε για την απεικόνιση, δηλαδή από ποια οπτική γωνία θέλουμε να δούμε το αποτέλεσμα. Τα μέτρα αυτά είναι ο Βαθμός Κεντρικότητας (Centrality Degree), η Παραπλήσια Κεντρικότητα (Closeness Centrality), ο Βαθμός Πρεστίζ (Prestige Degree), η Εγγύτητα Πρεστίζ (Proximity Prestige) και ο Συντελεστής Ομαδοποίησης (Clustering Coefficient). Ανάλογα με το ποιο μέτρο από τα παραπάνω θα επιλεγεί αλλάζει και το μέγεθος του κόμβου για να μπορούν να διαχωριστούν όσο είναι αυτοί που ξεχωρίζουν. Επίσης υπάρχει η δυνατότητα να εμφανίζονται με μεγαλύτερη λεπτομέρεια ή όχι οι ακμές, με βάση τα δευτερόλεπτα συνομιλίας μεταξύ των κόμβων και η δυνατότητα αλλαγής του χρώματος των κόμβων, εκτός από τον αρχικό κόμβο που παραμένει πάντα κόκκινος.

Σε μερικά δίκτυα παρατηρείται το φαινόμενο πολλών κόμβων με Out_Degree=1 οι οποίοι δημιουργούν αρκετό «θόρυβο» μέσα στην εικόνα και δυσκολεύουν την ανάλυση. Για να γίνει λίγο πιο ξεκάθαρη απεικόνιση υπάρχει η επιλογή της ενεργοποίησης ενός μαγνητικού πεδίου το οποίο ωθεί τους κόμβους αυτούς προς τα δεξιά της εικόνας. Με την μέθοδο αυτή επιτυγχάνεται ο διαχωρισμός των κόμβων και η καλύτερη ανάλυση σε γραφήματα μεγάλου μεγέθους.

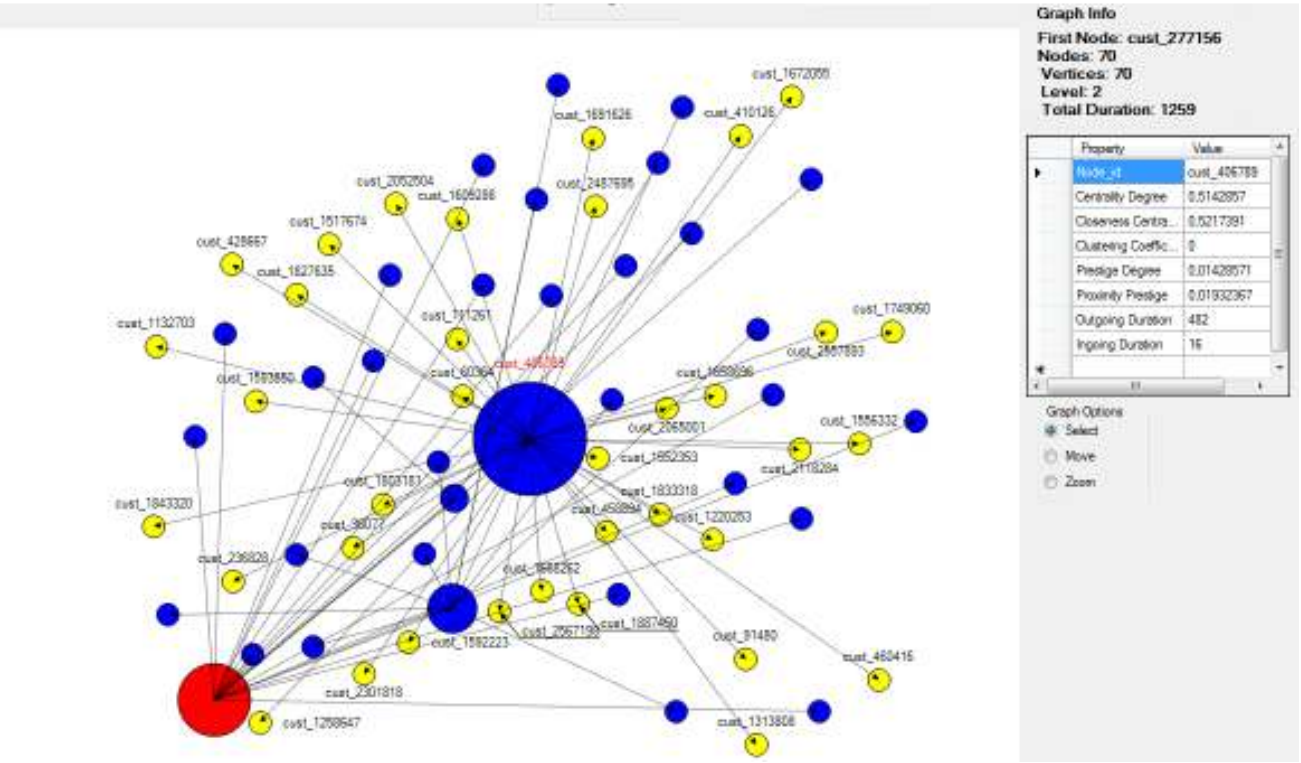
Μια ακόμα ρύθμιση για διαφορετική απεικόνιση αποτελεί η τοποθέτηση του αρχικού κόμβου σε ένα σταθερό σημείο και γύρω από αυτόν να προσαρμόζονται οι υπόλοιποι κόμβοι.

Ανάλυση Δικτύων

Στην ενότητα αυτή θα παρουσιάσω ορισμένα από τα αποτελέσματα της εφαρμογής και θα προσπαθήσω να δώσω την κατάλληλη ερμηνεία σε αυτά με βάση τα διαφορετικά μέτρα και τις εναλλακτικές απεικονίσεις. Έχω επιλέξει τέσσερις περιπτώσεις χρηστών που εμφανίζουν ετερογενή δίκτυα και παρουσιάζω τις απεικονίσεις αυτών μαζί με μια περιληπτική ανάλυση τους.

Περίπτωση 1

Αρχικός Κόμβος:cust_277156
Επίπεδο : 2
Σύνολο Κόμβων : 70
Σύνολο Ακμών : 70
Μέτρο: Βαθμός Κεντρικότητας (Degree Centrality)



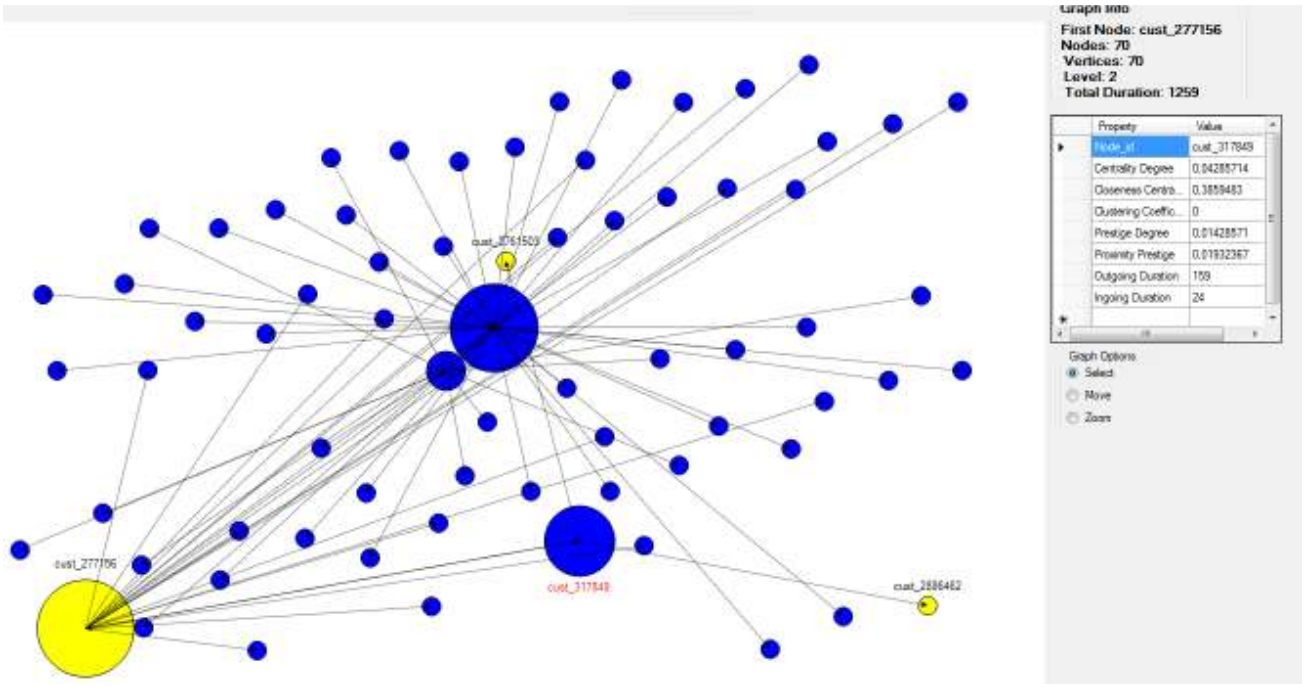
Απεικόνιση 1 1

Στο δίκτυο αυτό παρατηρούμε την ύπαρξη ενός κόμβου(cust_406789) με μεγαλύτερο βαθμό κεντρικότητας από τον αρχικό ,η διαφορά παρουσιάζεται με το μέγεθος του κόμβου και επίσης από την επιλογή του κόμβου (εμφανίζεται η ταυτότητα του με κόκκινα γράμματα) ενώ με κίτρινο χρώμα εμφανίζονται οι γείτονες αυτού του ισχυρού κόμβου. Η λέξη «ισχυρός» τον χαρακτηρίζει λόγω του μεγάλου βαθμού κεντρικότητας που έχει, το πλεονέκτημα αυτό του δίνει περισσότερες επιλογές και εναλλακτικές μέσα στο δίκτυο. Δηλαδή λόγω των αρκετών επιλογών που διαθέτει, άμεσα, είναι λιγότερο εξαρτημένος από ένα απομονωμένο κόμβο. (Hannenman, et al., 2005) Στην περίπτωση του τηλεπικοινωνιακού δικτύου η «δύναμη» αυτή μεταφράζεται ως ένας πελάτης που δεν

πρέπει να χαθεί ή αν δεν ανήκει στο δίκτυο θα μπορούσε να γίνει μία προσπάθεια προσέλκυσης του.

Ας εξετάσουμε τώρα το ίδιο δίκτυο αλλά με βάση άλλο μέτρο της Ανάλυσης Κοινωνικών Δικτύων, αυτό της Παραπλήσιας Κεντρικότητας (Closeness Centrality)

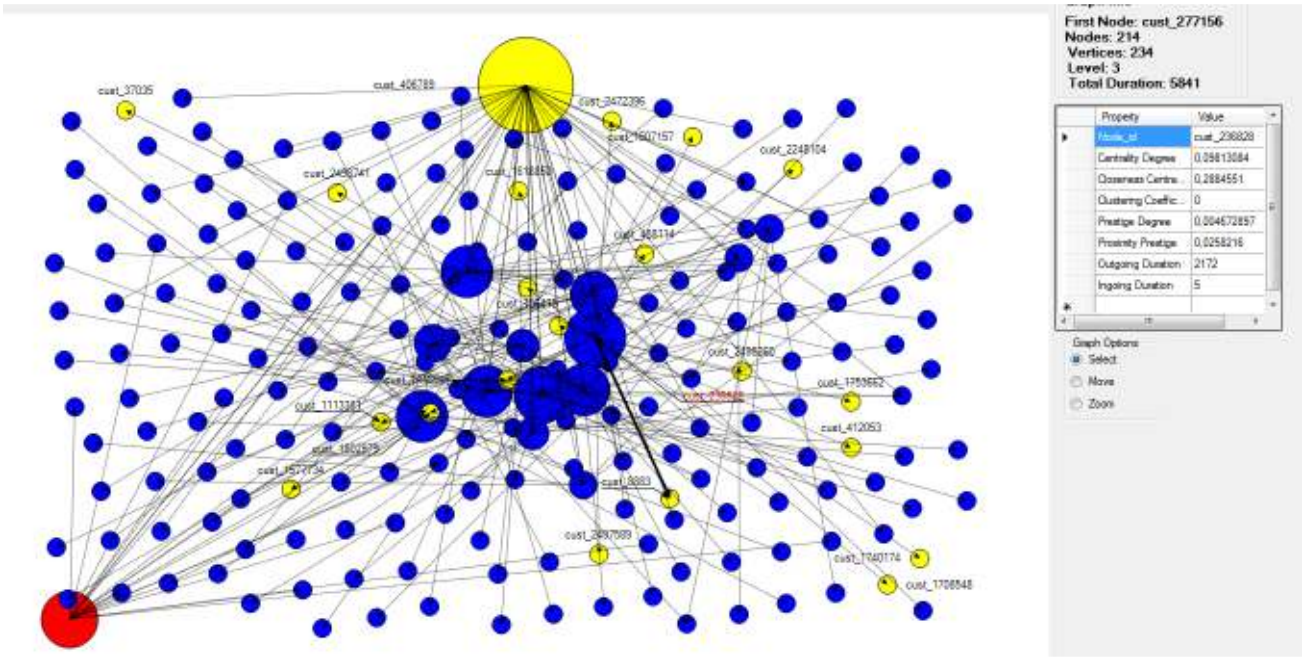
Αρχικός Κόμβος:cust_277156
Επίπεδο : 2
Σύνολο Κόμβων : 70
Σύνολο Ακμών : 70
Μέτρο: Παραπλήσια Κεντρικότητα (Closeness Centrality)



Απεικόνιση 1 2

Στην απεικόνιση αυτή το μέγεθος των κόμβων βασίζεται στην Παραπλήσια Κεντρικότητα, ένα ενδιαφέρον εύρημα είναι ο κόμβος που έχει επιλεχθεί, ο οποίος αν και είχε μικρό Βαθμό Κεντρικότητας παρουσιάζει την τρίτη μεγαλύτερη Παραπλήσια Κεντρικότητα. Άρα ο κόμβος αυτός αν και δεν έχει πολλές αρχικές επιλογές μπορεί να επικοινωνήσει γρήγορα με όλο το δίκτυο και να μοιράσει ή να δεχτεί πληροφορίες. Ο κόμβος αυτός παρουσιάζει τη συμπεριφορά αυτή γιατί είναι ο μόνος που επικοινωνεί με τον αρχικό και έτσι «εκμεταλλεύεται» το δίκτυο του Επιπλέον παρατηρούμε πως ο αρχικός κόμβος είναι αυτός με την μεγαλύτερη Παραπλήσια Κεντρικότητα, γεγονός που είναι φυσιολογικό λόγω της σχεδίασης του δικτύου έως το $2^{ο}$ επίπεδο του κόμβου αυτού. Τα αποτελέσματα αυτά θα μπορούσε να είναι διαφορετικά αν αλλάζαμε το αρχικό επίπεδο αλλά με τον ίδιο αρχικό κόμβο.

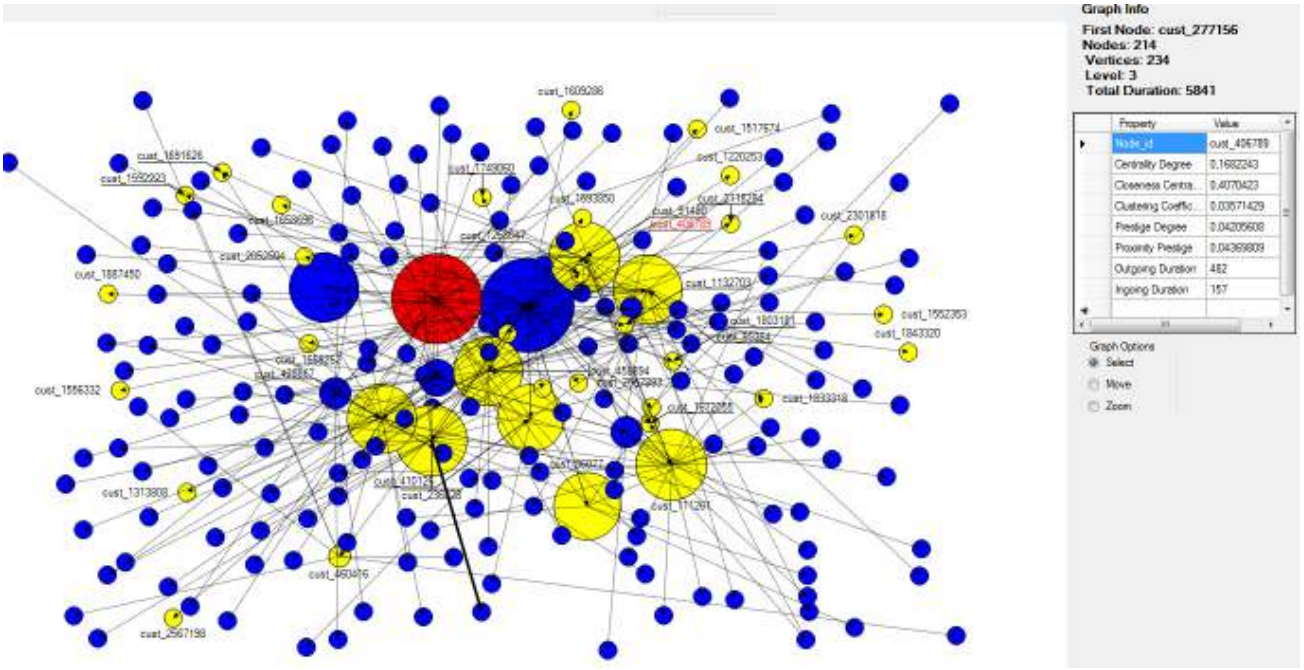
Αρχικός Κόμβος:cust_277156
Επίπεδο : 3
Σύνολο Κόμβων : 214
Σύνολο Ακμών : 234
Μέτρο: Βαθμός Κεντρικότητας (Degree Centrality)



Απεικόνιση 1 3

Η παραπάνω απεικόνιση είναι παρόμοια με την απεικόνιση 1 1 με μόνη διαφορά την αύξηση του επιπέδου. Η διαφορά αυτή όμως είναι αρκετή για να εμφανιστούν αρκετοί πιο «ισχυροί» κόμβοι από τον αρχικό, δηλαδή με μεγαλύτερο Βαθμό Κεντρικότητας. Εξίσου ενδιαφέροντα είναι τα αποτελέσματα στην απεικόνιση με βάση την Παραπλήσια Κεντρικότητα που ακολουθεί.

Αρχικός Κόμβος:cust_277156
Επίπεδο : 3
Σύνολο Κόμβων : 214
Σύνολο Ακμών : 234
Μέτρο: Παραπλήσια Κεντρικότητα (Closeness Centrality)



Απεικόνιση 1 4

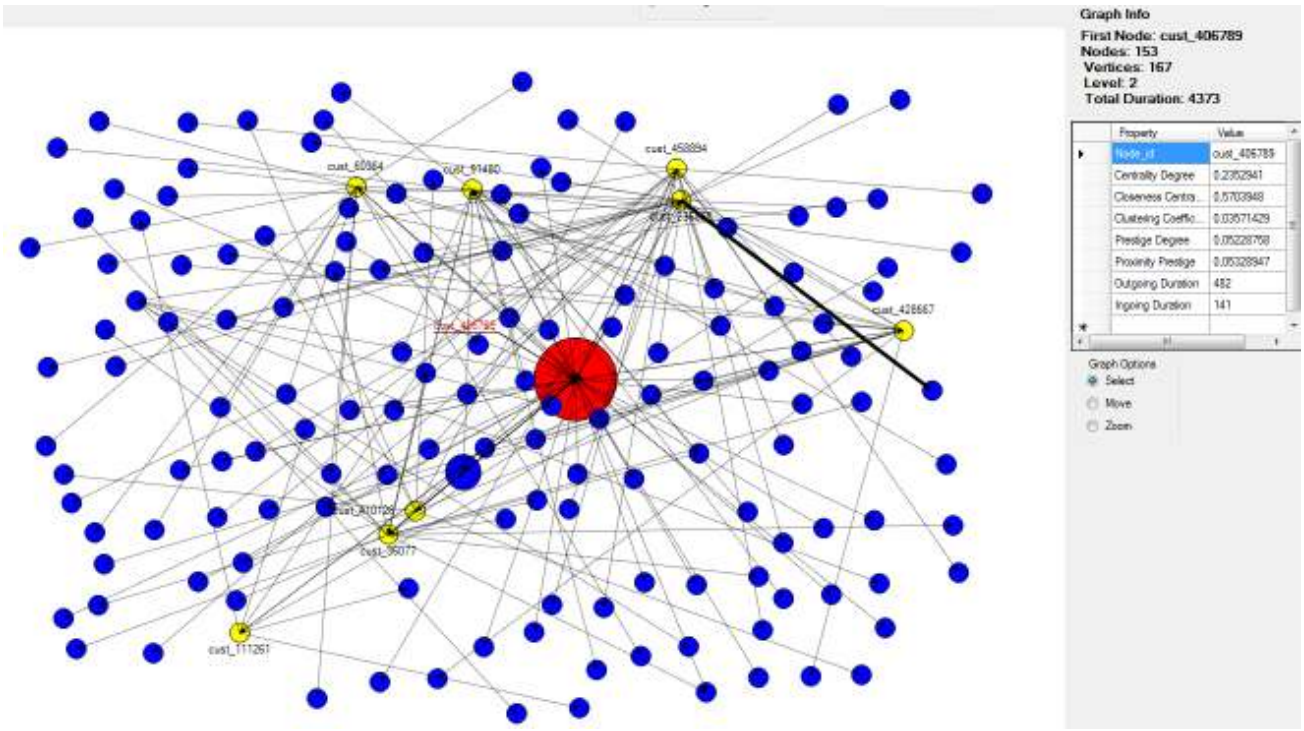
Εδώ πλέον ο κόμβος με την μεγαλύτερη Παραπλήσια Κεντρικότητα είναι ο cust_406789 λόγω της αύξησης του επιπέδου και τη δυνατότητα του κόμβου αυτού να επικοινωνεί γρηγορότερα μέσα σε όλο το δίκτυο. Λόγω των χαρακτηριστικών που παρουσίασε ο κόμβος αποτελεί τον επόμενο κόμβο προς ανάλυση.

Ένας πίνακας που συνοψίζει τα παραπάνω συμπεράσματα και αποτελεί μία σύγκριση του Βαθμού Κεντρικότητας (Centrality Degree) και της Παραπλήσιας Κεντρικότητας (Closeness Centrality) είναι ο παρακάτω:

	Χαμηλός Βαθμός Κεντρικότητας	**Χαμηλή Παραπλήσια Κεντρικότητα**
Υψηλός Βαθμός Κεντρικότητας		Κόμβος που βρίσκεται σε ένα σύμπλεγμα μακρυά από το υπόλοιπο δίκτυο
Υψηλή Παραπλήσια Κεντρικότητα	Κόμβος που είναι συνδεδεμένος με «ισχυρούς»/ «σημαντικούς» κόμβους του δικτύου	

Περίπτωση 2

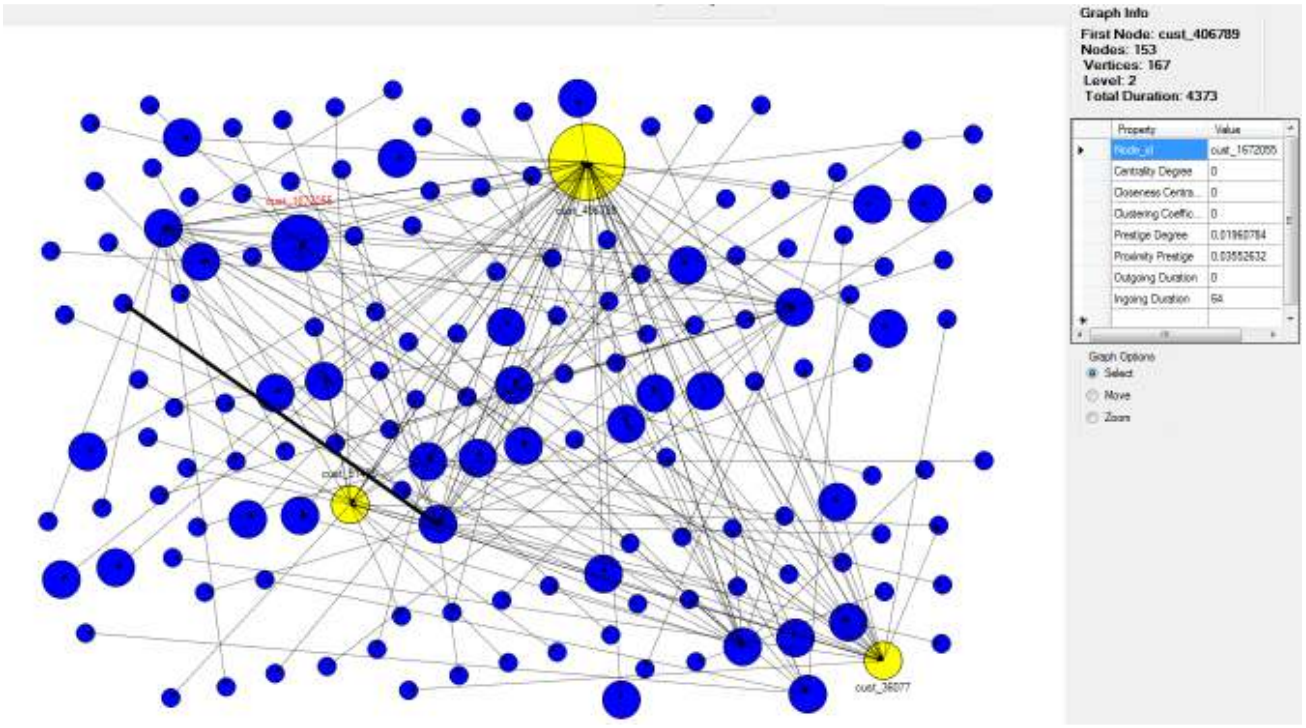
Αρχικός Κόμβος:cust_406789
Επίπεδο : 2
Σύνολο Κόμβων : 153
Σύνολο Ακμών : 167
Μέτρο: Βαθμός Πρεστίζ (Prestige Degree)



Απεικόνιση 2 1

Στην παραπάνω απεικόνιση χρησιμοποιείται το μέτρο Βαθμός Πρεστίζ (Prestige Degree), όπου παρατηρείται πως ο αρχικός κόμβος(χρωματισμένος με κόκκινο) είναι αυτός που δέχεται τις περισσότερες συνδέσεις(με κίτρινο χρώμα) μέσα στο δίκτυο. Άρα είναι αυτός που δέχεται το μεγαλύτερο μέγεθος πληροφοριών και απολαμβάνει ένα είδος υποστήριξης γιατί σε αυτόν απευθύνονται αρκετοί κόμβοι. Όμως στην παραπάνω απεικόνιση παρατηρούμε και ένα μειονέκτημα του μέτρου αυτού, ότι υπάρχει μόνο ένας κόμβος με μεγάλο Βαθμό Πρεστίζ. Αυτό είναι ένα φαινόμενο που δύσκολα ανταποκρίνεται στην πραγματικότητα και οφείλουμε να μελετήσουμε και ένα ακόμα μέτρο Πρεστίζ για να δούμε την πραγματική εικόνα.

Αρχικός Κόμβος:cust_406789
Επίπεδο : 2
Σύνολο Κόμβων : 153
Σύνολο Ακμών : 167
Μέτρο: Εγγύτητα Πρεστίζ(Proximity Prestige)



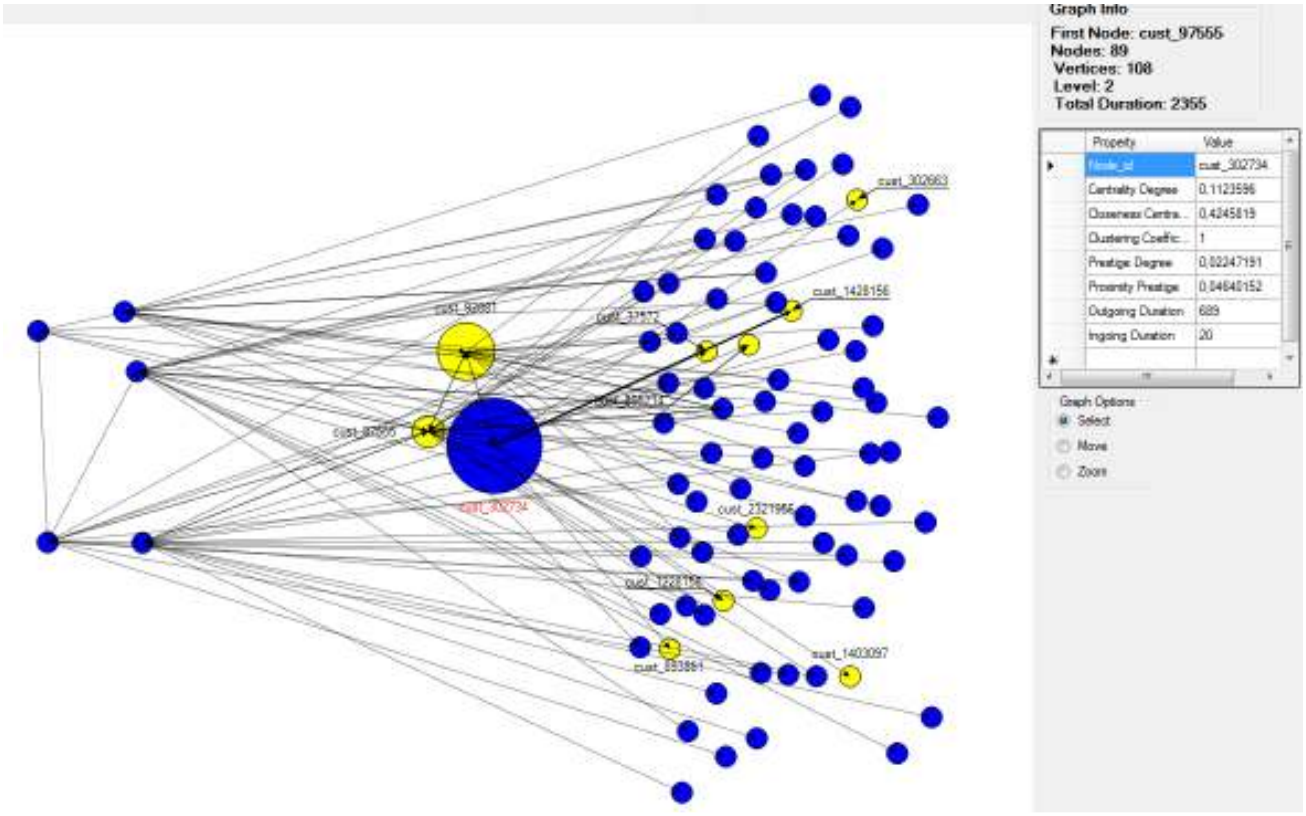
Απεικόνιση 2 2

Εδώ παρατηρούμε την εμφάνιση αρκετών κόμβων με σημαντική Εγγύτητα Πρεστίζ (ProximityPrestige) που δεν παρουσιαζόταν στο προηγούμενο παράδειγμα. Άρα υπάρχουν κόμβοι που ενώ δεν έχουν αρχικά μεγάλη υποστήριξη σε πρώτο επίπεδο μέσω του δικτύου μπορούν να λάβουν γρήγορα πληροφορίες λόγω τις μικρής του απόστασης από το μέσο όρο των κόμβων του δικτύου. Αυτό δικαιολογείται από το γεγονός ότι υπάρχουν αρκετοί κόμβοι με σημαντικό Βαθμό Πρεστίζ (Prestige Degree) και έτσι διαχέεται η πληροφορία μέσω αυτών. Ένας πίνακας που κάνει μια σύγκριση των αποτελεσμάτων ανάμεσα σε Βαθμό Πρεστίζ(Prestige Degree) και Εγγύτητα Πρεστίζ(Proximity Prestige) είναι ο παρακάτω και παρουσιάζει μία αντιστοιχία με αυτόν του Βαθμού Κεντρικότητας (Centrality Degree) που περιγράφθηκε παραπάνω:

	Χαμηλός Βαθμός Πρεστίζ	Χαμηλή Εγγύτητα Πρεστίζ
Υψηλός Βαθμός Πρεστίζ		Κόμβος που βρίσκεται σε ένα σύμπλεγμα που δεν δέχεταιι πολλές πληροφορίες
Υψηλή Εγγύτητα Πρεστίζ	Κόμβος που είναι συνδεδεμένος με κόμβους που έχουν μεγάλη υποστήριξη	

Περίπτωση 3

Αρχικός Κόμβος:cust_97555
Επίπεδο : 2
Σύνολο Κόμβων : 89
Σύνολο Ακμών : 108
Μέτρο: Συντελεστής Ομαδοποίησης(Clustering Coefficient)

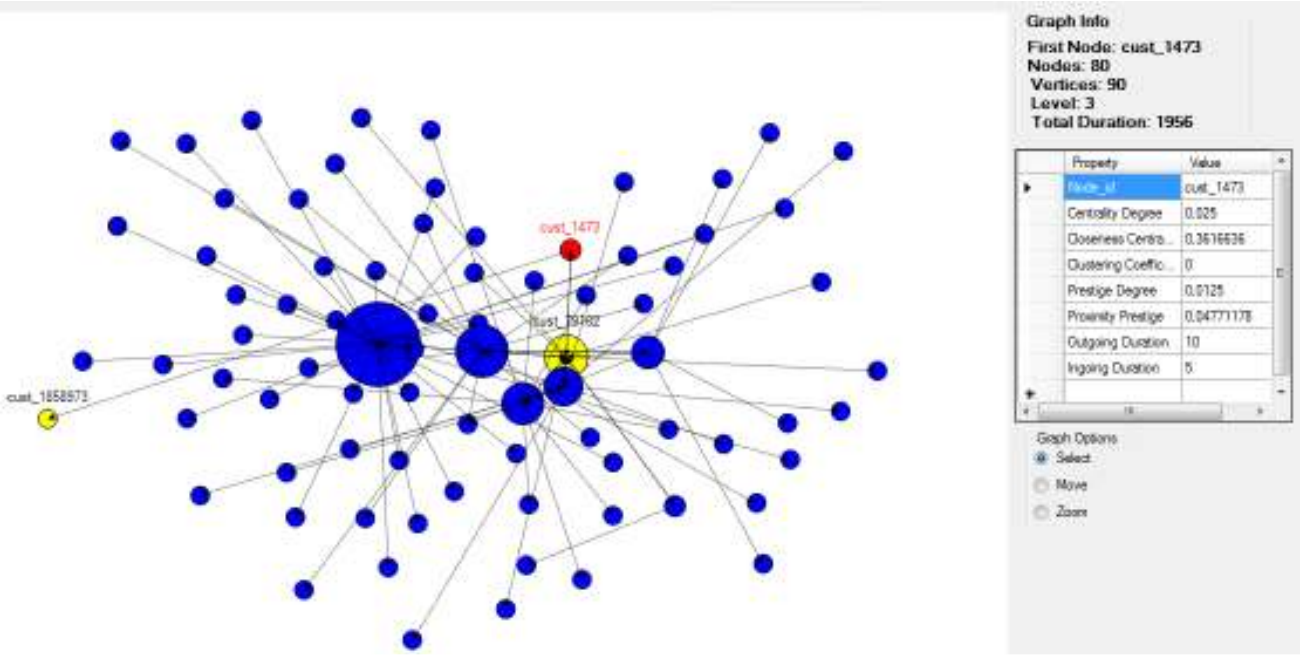


Ο Συντελεστής Ομαδοποίησης(Clustering Coefficient) μας δίνει την πιθανότητα να ανήκει ένα κόμβος σε μία κλίκα, δηλαδή σε ένα κλειστό δρόμο όπου όλοι οι κόμβοι επικοινωνούν μεταξύ τους. Ο επιλεγμένος κόμβος (cust_302734)έχει συντελεστή ομαδοποίησης(Clustering Coefficient) ίσο με ένα, συνεπώς ανήκει σίγουρα σε μία κλίκα και αυτή μπορεί να παρατηρηθεί με τους δύο κόμβους που είναι δίπλα του και είναι

χρωματισμένοι με κίτρινο χρώμα. Αντίστοιχα οι πιθανότητες του ενός από αυτούς είναι 0.5, δηλαδή ανήκει σχεδόν σε μία κλίκα και για να επιτευχθεί αυτό θα έπρεπε να επικοινωνούν περισσότεροι κόμβοι της γειτονιάς του μεταξύ τους.

Περίπτωση 4

Αρχικός Κόμβος:cust_1473
Επίπεδο : 3
Σύνολο Κόμβων : 80
Σύνολο Ακμών : 90
Μέτρο: Βαθμός Κεντρικότητας(Centrality Degree)



Στο παραπάνω δίκτυο έχουμε επιλέξει ως αρχικό κόμβο τον cust_1473 ο οποίος έχει μόνο δύο εξωτερικές επαφές, άρα από μία αρχική ανάλυση μόνο των εξωτερικών κόμβων δεν θα μπορούσε να προκύψει κάτι. Όμως στο δίκτυο του μέχρι το επίπεδο 3 εμφανίζονται 5 ισχυροί χρήστες οι οποίοι στη συνέχεια δημιουργούν ένα αρκετά μεγάλο δίκτυο. Το ενδιαφέρον στην περίπτωση αυτή είναι πως προκύπτει ένα μεγάλο δίκτυο από ένα κόμβο με μόνο δύο εξωτερικές συνδέσεις. Αυτό αποδεικνύει τη χρησιμότητα της Ανάλυσης Κοινωνικών Δικτύων, διότι αρχικά έχουμε ένα «αδύνατο» με βάση τις συνδέσεις τους όμως αυτός ο χρήστης δημιουργεί ένα μεγάλο δίκτυο το οποίο είναι σημαντικό για να ερευνηθεί.

Επεκτάσεις Εφαρμογής

Η παραπάνω εφαρμογή αποτελεί μια βάση για την Ανάλυση Κοινωνικών Δικτύων (Social Network Analysis, SNA) και σχεδιάστηκε με ένα συγκεκριμένο σκοπό σύμφωνα με τις ανάγκες ενός τηλεπικοινωνιακού δικτύου. Παρόλα αυτά υπάρχει η ευελιξία της μετατροπής της ανάλογα με τις προϋποθέσεις που υπάρχουν για το δίκτυο και για την Ανάλυση αυτού. Με την προοπτική αυτή προτείνονται ορισμένες επεκτάσεις της εφαρμογής για περαιτέρω χρήση αυτής.

Σε πρώτη φάση ο τρόπος ανάκτησης των δεδομένων είναι αρκετά απλός και ανταποκρίνεται πλήρως στη μορφή των κοινωνικών δικτύων. Συνεπώς η αλλαγή των αντικειμένων που έχουν δημιουργηθεί δεν είναι αναγκαία, το μόνο που καθίσταται απαραίτητο είναι η δημιουργία ενός πίνακα στη βάση δεδομένων στη μορφή που έχει περιγραφεί παραπάνω.

Μέτρα Ανάλυσης Κοινωνικών Δικτύων

Όσον αφορά τα μέτρα της Ανάλυσης Κοινωνικών Δικτύων (Social Network Analysis,SNA) υπάρχει η δυνατότητα επιλογής περισσοτέρων. Ένα προτεινόμενο μέτρο που έχει περιγραφεί και στο Θεωρητικό Μέρος είναι το Bettweenness Centrality, το οποίο αποτελεί ένα μέτρο που προσδίδει τη θέση του χρήστη μέσα στο δίκτυο ανάλογα με τον αριθμό των γεωδαιτικών που διέρχονται από αυτό. Όσο περισσότερες γεωδαιτικές διέρχονται από ένα κόμβο τόσο ισχυρότερη θέση αποκτά ο κόμβος αυτό μέσα στο δίκτυο γιατί αποτελεί αναγκαίο σύνδεσμο μέσα στις γεωδαιτικές.

Επιπλέον για τα μέτρα Κεντρικότητας προτείνεται και η Κεντρικότητα Ιδιοδιανυσμάτων (Information Centrality) το οποίο υπολογίζει τις ιδιοτιμές και τα ιδιοδιανύσματα του Πίνακα Επικοινωνίας του Δικτύου και το ιδιοδιάνυσμα που αντιστοιχεί στην μεγαλύτερη ιδιοτιμή αποτελεί του βαθμού Κεντρικότητας των κόμβων. Το μέτρο αυτό εστιάζει στον τρόπο με τον οποίο η πληροφορία μπορεί να κυλίσει μέσω των διαφορετικών μονοπατιών σύμφωνα με το βάρος των ακμών και τις γεωδαιτικές.

Αντίστοιχο μέτρο με την Κεντρικότητα Ιδιοδιανυσμάτων (Information Centrality) αποτελεί και η Τάξη ή Στάτους Πρεστίζ (Rank or Status Prestige) και υπολογίζεται με περίπου την ίδια διαδικασία, υπολογίζοντας ιδιοτιμές και ιδιοδιανύσματα. Η Τάξη ή Στάτους Πρεστίζ (Rank or Status Prestige) βασίζεται επίσης στις θέσεις των γειτόνων και στο πόσο σημαντικοί είναι αυτή για να προσδώσουν μεγάλο πρεστίζ στον ίδιο τον κόμβο.

Αλγόριθμος Απεικόνισης

Ο αλγόριθμος που χρησιμοποιώ για την απεικόνιση του δικτύου και η μέθοδος «Follow Your Nose» δύναται να χρησιμοποιηθεί έως ένα περιορισμένο μέγεθος δικτύου. Η υπολογιστική ικανότητα του αλγορίθμου που χρησιμοποιώ ανέρχεται σε δίκτυα μεγέθους έως 500 κόμβους. Συνεπώς για την απεικόνιση δικτύων μεγαλύτερου μεγέθους μπορεί να βελτιωθεί ο αλγόριθμος αυτός ή να χρησιμοποιηθεί ένας διαφορετικός με μεγαλύτερες δυνατότητες (Kobourov, 2012).

Οι αλγόριθμοι που προτείνονται στο παραπάνω βιβλίο ανήκουν επίσης στην κλάση των force-directed αλλά δεν βασίζονται στη μέθοδο «Follow Your Nose» και χρησιμοποιούν τεχνικές τμηματικού υπολογισμού των δυνάμεων σε κάθε σύμπλεγμα. Επιπλέον δεν ορίζεται ένας σταθερός αριθμός επαναλήψεων αλλά τρέχουν έως ότου επιτευχθεί η ανάλογη ισορροπία στο σύστημα δυνάμεων. Υπάρχουν αλγόριθμοι με δυνατότητα

απεικόνισης αρκετών χιλιάδων κόμβων και μπορούν να χρησιμοποιηθούν στην υπάρχουσα βάση της εφαρμογής με σύντομες αλλαγές στον κώδικα και τις μεθόδους.

Εναλλακτικές Απεικονίσεις

Οι επιλογές που δίνονται για τις απεικονίσεις στην εφαρμογή στην ουσία είναι τρεις, αυτή του σταθερού αρχικού κόμβου, αυτή με το μαγνητικό πεδίο και τέλος η απουσία του αρχικού κόμβου.

Μία πρόταση για εμπλουτισμό των απεικονίσεων αυτό αποτελούν οι σταθερές απεικονίσεις με διαφορετικά σχήματα, όπως αυτό του τετραγώνου, του κύκλου και άλλων. Δηλαδή θα μπορούσε να μένουν σταθεροί ορισμένοι κόμβοι στις άκρες ενός γεωμετρικού σχήματος και στη συνέχεια να υπολογίζονται οι δυνάμεις των υπολοίπων κόμβων στο εσωτερικό του σχήματος αυτού. Με τον τρόπο αυτό μπορούμε να απομονώσουμε ορισμένους κόμβους και να επιτευχθούν πιο ξεκάθαρα αποτελέσματα.

Όσον αφορά το μαγνητικό πεδίο μπορεί να τροποποιηθεί ανάλογα με τα επιθυμητά αποτελέσματα. Στην εφαρμογή αυτή με το μαγνητικό πεδίο απομονώνουμε τους κόμβους με αριθμό εσωτερικών συνδέσεων ένα, ο περιορισμός αυτός μπορεί να μεταβληθεί και να προσαρμόζεται στην εικόνα που προσπαθούμε να εστιάσουμε. Εκτός από τον συγκεκριμένο τύπο μαγνητικού πεδίου υπάρχουν και άλλα που κατευθύνουν τους κόμβους στο κέντρο της απεικόνισης ή ακόμα και προς τα τοιχώματα.

Αποτίμηση

Κατά τη διάρκεια της διπλωματικής μου αποκόμισα σημαντικές γνώσεις για τον προγραμματισμό, τις βάσεις δεδομένων, γνώρισα ένα νέο πολύ ενδιαφέρον θέμα, την Ανάλυση Κοινωνικών Δικτύων (Social Network Analysis) και εμπλούτισα τις γνώσεις μου πάνω στη θεωρία γραφημάτων, με τους αλγόριθμους της κλάσης Force-directed.

Αρχικά μέσα από την πρακτική μου άσκηση στην εταιρεία Datamine απέκτησα μία πρώτη επαφή με τις βάσεις δεδομένων και τον αντικειμενοστραφή προγραμματισμό μέσω της γλώσσας C#. Η ενασχόληση μου στην εταιρεία αυτή περιελάμβανε την επεξεργασία, δημιουργία πινάκων στις βάσεις δεδομένων και χρησιμοποίηση της σύνδεσης μεταξύ των βάσεων δεδομένων και της C# μέσω σχεδιασμού τετριμμένων εφαρμογών.

Στην συνέχεια ασχολήθηκα με την Ανάλυση Κοινωνικών Δικτύων (Social Network Analysis, SNA), διάβασα για το σκοπό αυτής και μελέτησα τα μέτρα της. Από την Βιβλιογραφία επέλεξα ορισμένα μέτρα της Ανάλυσης Κοινωνικών Δικτύων που ταίριαζαν με τον σκοπό της διπλωματικής και συνέταξα το θεωρητικό μέρος της εργασίας. Με βάση τα μέτρα αυτά προχώρησε ο σχεδιασμός της εφαρμογής.

Η τελική εικόνα και λειτουργία της εφαρμογής, των μεθόδων της και των αντικειμένων της διαφέρει αρκετά από τον αρχικό σχεδιασμό. Γεγονός που οφείλεται στην συνεχή βελτίωση και μετατροπή των μεθόδων και αντικειμένων, λόγω της αρχικής που απειρίας στον προγραμματισμό. Μέσα από την υλοποίηση της μάθαινα καθημερινά νέους τρόπους προγραμματισμού και προσπαθούσα για την βελτιστοποίηση των μεθόδων. Για παράδειγμα ,στο πρώτο κομμάτι της εφαρμογής που γίνεται η σύνδεση με την βάση δεδομένων αρχικά χρησιμοποιούσα για κάθε επίπεδο διαφορετικό query για την ανάκτηση της πληροφορίας, δηλαδή μία τετριμμένη λύση, αλλά στη συνέχεια σχεδίασα ένα δυναμικό query το οποίο ανάλογα με το επίπεδο χτίζεται δυναμικά και φέρει το επιθυμητό αποτέλεσμα.

Στο μέρος της εφαρμογής όπου ασχολήθηκα με το σχεδιασμό του γραφήματος, η πρώτη απόπειρα μου έγινε δίχως να έχω διαβάσει για του αλγόριθμους Force-directed και κρίθηκε αποτυχημένη. Στη συνέχεια ύστερα από ενδελεχή μελέτη περί των αλγορίθμων Force-directed ξεκίνησα την σύνταξη των μεθόδων για την απεικόνιση του γραφήματος χρησιμοποιώντας μία έτοιμη βιβλιοθήκη της C# το MS Chart Control. Το τελικό αποτέλεσμα προέκυψε ύστερα από αρκετές δοκιμές στις παραμέτρους του αλγορίθμου και στον τρόπο απεικόνισης των κόμβων.

Συνοψίζοντας, λόγω της καθημερινής μου τριβής με την παραπάνω εφαρμογή για αρκετούς μήνες πιστεύω πως οι γνώσεις που απέκτησα και τα εφόδια από την εργασία αυτή θα με βοηθήσουν στην συνέχεια των σπουδών μου ή της επαγγελματικής μου σταδιοδρομίας. Διότι γνώρισα και ασχολήθηκα με καινούρια θέματα που δεν περιλαμβάνονταν στο πρόγραμμα σπουδών της Σχολής μου και πλέον πιστεύω πως μπορώ να ασχοληθώ στη συνέχεια με τα παραπάνω.

Βιβλιογραφία

Force-directed Algorithms [Ενότητα βιβλίου] / συγγρ. Kobourov S. G. // Handbook of Graph Drawing and Visualization / συγγρ. βιβλίου Tamasia Roberto. - 2012.

Graph Drawing Algorithms for the Visualization of Graphs [Βιβλίο] / συγγρ. Di Battista Giuseppe [και συν.]. - 1999.

Graph Drawing by Force-directed Placement [Ενότητα βιβλίου] / συγγρ. Fruchterman M.,Thomas και Reingold.M. Edward // Software-Practice and Experience, Vol.21. - 1991.

Introduction to Social Network Methods [Βιβλίο] / συγγρ. Hannenman Robert και Riddle Mark. - 2005.

Node Centrality in Weighted Networks: Generalizing Degree and Shortest Paths [Επιθεώρηση] / συγγρ. Opsahl Tore, Agneessems Filip και Skvoretz John. - 2010.

Programming C# 4.0 [Βιβλίο] / συγγρ. Griffiths Ian, Adams Mathews και Liberty Jesse. - 2010.

Sams Teach Yourself SQL in 24 Hours [Βιβλίο] / συγγρ. Steohens Ryan, Plew Ron και Jones D.,Arie. - 2008.

Social Network Analysis a Handbook [Βιβλίο] / συγγρ. Scott John. - London : [s.n.], 1987.

Social Network Analysis Methods and Applications [Βιβλίο] / συγγρ. Wasserman Stanley και Faust Katherine. - 1994.

Θεωρία Γραφημάτων [Βιβλίο] / συγγρ. Παπαϊωάννου Αλέξανδρος. - Αθήνα : [s.n.], 2006.

Σχεδιασμός Αλγορίθμων [Βιβλίο] / συγγρ. Kleinberg Jon και Tardos Eva. - 2006.

Παράρτημα

Στο παράρτημα παρουσιάζεται η αρχιτεκτονική της εφαρμογής και οι μέθοδοι που χρησιμοποιούνται με μικρή επεξήγηση πάνω στην λειτουργία τους. Η εφαρμογή αυτή βασίζεται πάνω σε τρία πρότζεκτ που συνδυάζονται στο τελικό GUI της εφαρμογής. Ακολουθεί η περιγραφή του κάθε πρότζεκτ και ο κώδικας που περιλαμβάνουν.

DAL project

Στο πρότζεκτ αυτό ανήκει η κλάση DBManager στην οποία γίνεται η σύνδεση με την βάση δεδομένων, η ανάκτηση των δεδομένων και η αποθήκευση τους σε αντικείμενα της C#.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;
using Entities;

namespace DAL
{
publicclassDBManager
    {
privatestring strConn;
public DBManager()
        {
strConn = @"Data Source=.\;" + "Initial Catalog=Telecom; Integrated Security=True;";
        }
privateSqlConnection cn;

privatevoid OpenConn()
        {
if (cn == null)
cn = newSqlConnection(strConn);

cn.Open();
        }

privatevoid CloseConn()
        {
cn.Close();
        }

privatevoid CreateSqlForLevel(int currentLevel, int maxlevel, StringBuilder sb, string str)
        {
if (maxlevel != currentLevel)
            {
if (currentLevel == 0)
                {
str = "select anum,bnum,call_dur from final where anum in (@num)";
sb.Append(str);
CreateSqlForLevel(currentLevel + 1, maxlevel, sb, str);
                }
else
                {
str = str.Replace("@num", "select bnum from final where anum in (@num)");
```

```
sb.Append(" union ").Append(str);
CreateSqlForLevel(currentLevel + 1, maxlevel, sb, str);
                }
            }
        }

publicList<string> GetA()
        {
List<string> list = newList<string>();
string strSQL;
OpenConn();
strSQL = "select distinct top 1000 ANUM from final";
SqlCommand cmd = newSqlCommand(strSQL, cn);
SqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())
            {
string a = string.Empty;
                a = rdr.GetString(rdr.GetOrdinal("ANUM"));
list.Add(a);

            }
rdr.Close();
CloseConn();

return list;

        }
publicGraph GetTraffic(string anum, byte level)
        {
Graph result = newGraph();
StringBuilder sb = newStringBuilder();
CreateSqlForLevel(0, level, sb, "");
string strSQL = sb.Replace("@num", "'" + anum + "'").ToString();

OpenConn();
SqlCommand cmd = newSqlCommand(strSQL, cn);
            cmd.CommandTimeout = 1000;
SqlDataReader rdr = cmd.ExecuteReader();
while (rdr.Read())
            {
Node n1 = newNode();
Node n2 = newNode();
                n1.id = rdr.GetString(rdr.GetOrdinal("anum"));
                n2.id = rdr.GetString(rdr.GetOrdinal("bnum"));
int dur = rdr.GetInt32(rdr.GetOrdinal("call_dur"));
if (dur == 0)
dur = 5;
Node searcha = result.nodes.Find(x => x.id == n1.id);
if (searcha == null)
                {
Adjacency adj1 = newAdjacency();
                    adj1.id = n2.id;
                    adj1.dur = dur;
                    n1.Out_Adjacency.Add(adj1);
result.nodes.Add(n1);
                }
else
                {
Node f = result.nodes.Find(x => x.id == n1.id);
Adjacency exist_adj = f.Out_Adjacency.Find(x => x.id == n2.id);
if (exist_adj == null)
```

```
                    {
Adjacency new_adj = newAdjacency();
                        new_adj.id = n2.id;
                        new_adj.dur = dur;
                        f.Out_Adjacency.Add(new_adj);
                    }
else
                        exist_adj.dur += dur;

                }
Node searchb = result.nodes.Find(x => x.id == n2.id);
if (searchb == null)
                {
Adjacency adj2 = newAdjacency();
                    adj2.id = n1.id;
                    adj2.dur = dur;
                    n2.In_Adjacency.Add(adj2);
result.nodes.Add(n2);
                }
else
                {
Node f = result.nodes.Find(x => x.id == n2.id);
Adjacency exist_adj = f.In_Adjacency.Find(x => x.id == n1.id);
if (exist_adj == null)
                    {
Adjacency new_adj = newAdjacency();
                        new_adj.id = n1.id;
                        new_adj.dur = dur;
                        f.In_Adjacency.Add(new_adj);
                    }
else
                    {
                        exist_adj.dur += dur;
                    }
                }
                result.total_dur += dur;

            }
return result;
}
}
}
```

Entities project

Στο πρότζεκτ αυτό ανήκουν τρεις κλάσεις, η Adjacency, η Graph και η Node, οι οποίες αποτελούν τα νέα αντικείμενα που ορίζονται στην εφαρμογή και αποθηκεύεται το γράφημα.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Entities
{
publicclassAdjacency
    {
public Adjacency()
        {
this.dur = newint();
        }
```

```
publicstring id { get; set; }
publicint dur { get; set; }
}

}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Entities
{
publicclassGraph
    {
public Graph()
        {
this.nodes = newList<Node>();
this.total_dur = newint();
        }
publicList<Node> nodes { get; set; }
publicint total_dur { get; set; }
    }

publicclassOutput
    {
public Output()
        {
this.nodes = newList<Node>();
this.cent_deg = newList<Degree>();
this.prest_deg = newList<Degree>();
this.clust_coef = newList<Degree>();
this.clos_cent = newList<Degree>();
this.prox_prest = newList<Degree>();
this.total_dur = newint();
        }

publicList<Node> nodes { get; set; }
publicList<Degree> cent_deg { get; set; }
publicList<Degree> prest_deg { get; set; }
publicList<Degree> clust_coef { get; set; }
publicList<Degree> clos_cent { get; set; }
publicList<Degree> prox_prest { get; set; }
publicint total_dur { get; set; }
}
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Entities
{
publicclassNode
    {
public Node()
        {
this.Out_Adjacency = newList<Adjacency>();
this.In_Adjacency = newList<Adjacency>();
        }
```

```
publicstring id { get; set; }
publicList<Adjacency> Out_Adjacency { get; set; }
publicList<Adjacency> In_Adjacency { get; set; }
publicbyte level;
    }

publicclassDegree
    {
public Degree()
        {
this.deg = newfloat();
        }
publicstring id { get; set; }
publicfloat deg { get; set; }
    }

publicclassGraphNode
    {
public GraphNode()
        {
this.Out_Adjacency = newList<Adjacency>();
this.In_Adjacency = newList<Adjacency>();
this.level = newbyte();
this.X = newfloat();
this.Y = newfloat();
this.force_x = newfloat();
this.force_y = newfloat();
        }
publicstring id { get; set; }
publicList<Adjacency> Out_Adjacency { get; set; }
publicList<Adjacency> In_Adjacency { get; set; }
publicbyte level;
publicfloat X;
publicfloat Y;
publicfloat force_x;
publicfloat force_y;
publicint size;
}

}
```

GUI project

Στο πρότζεκτ αυτό ανήκουν δυο κλάσεις, η Metrics και η Graph-GUI. Η πρώτη είναι υπεύθυνη για τον υπολογισμό των μέτρων SNA που έχουν επιλεχθεί από την βιβλιογραφία και η δεύτερη για την δημιουργία του WindowsForm της εφαρμογής και το σχεδιασμό του γραφήματος.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Entities;
using DAL;

namespace GUI
{
classMetrics
    {
publicGraph GetNetwork(string anum, byte level)
```

```
        {
DBManager db = newDBManager();
Graph res = db.GetTraffic(anum, level);
return res;
        }

publicDictionary<String, List<string>> Favourite(Graph g)
        {
Dictionary<String, List<string>> result = newDictionary<String, List<string>>();
foreach (Node n in g.nodes)
            {
List<String> fav =
                    (from adj in n.Out_Adjacency
orderby adj.dur descending
select adj.id).ToList();
result.Add(n.id, fav);
            }
return result;
        }

publicOutput Cent_Prest_Degrees(Graph g)
        {
Output result = newOutput();
List<Degree> unsorted_centrality = newList<Degree>();
List<Degree> unsorted_prestige = newList<Degree>();
foreach (Node n in g.nodes)
            {
int deg1 = n.Out_Adjacency.Count();
if (deg1 != 0)
                {
Degree d1 = newDegree();
                    d1.deg = (float)deg1 / g.nodes.Count;
                    d1.id = n.id;
                    unsorted_centrality.Add(d1);
                }
int deg2 = n.In_Adjacency.Count();
if (deg2 != 0)
                {
Degree d2 = newDegree();
                    d2.deg = (float)deg2 / g.nodes.Count();
                    d2.id = n.id;
                    unsorted_prestige.Add(d2);
                }
            }
List<Degree> centrality = SortDegree(unsorted_centrality);
List<Degree> prestige = SortDegree(unsorted_prestige);
            result.cent_deg = centrality;
            result.prest_deg = prestige;
return result;
        }

publicList<Degree> ClusteringCoefficiant(Graph g)
        {
List<Degree> unsorted_result = newList<Degree>();
foreach (Node n in g.nodes)
            {
List<string> neighboorhood = newList<string>();
int k = 0;
int e = 0;
foreach (Adjacency ad in n.Out_Adjacency)
                {
var check = g.nodes.Find(x => x.id == ad.id);
```

```
if (check != null)
                    {
if (check.Out_Adjacency.Find(x => x.id == n.id) != null)
                        {
                            k += 1;
neighboorhood.Add(ad.id);
                        }
                    }
                }
if (k > 1)
                {
foreach (string neigh in neighboorhood)
                    {
var check = g.nodes.Find(x => x.id == neigh);
if (check != null)
                        {
foreach (string nei in neighboorhood)
                            {
if (check.Out_Adjacency.Find(x => x.id == nei) != null)
                                {
                                    e += 1;
                                }
                            }
                        }
                    }
                }
if (e != 0)
                {
Degree d = newDegree();
                    d.id = n.id;
                    d.deg = (float)e / (k * (k - 1));
                    unsorted_result.Add(d);
                }
            }
List<Degree> result = SortDegree(unsorted_result);
return result;
        }

publicOutput Clos_Cent_Prox_Prest(Graph g)
        {
Output result = newOutput();
List<Degree> unsorted_closeness = newList<Degree>();
List<Degree> unsorted_proximity = newList<Degree>();
foreach (Node n in g.nodes)
            {
Degree deg = newDegree();
                deg.id = n.id;
Dictionary<string, int> dist = GetGeod(g, n, true);
int d = 0;
if (dist.Count != 0)
                {
foreach (var geo in dist)
{
                        d += (int)geo.Value;
                    }
                    deg.deg = (float)(dist.Keys.Count * dist.Keys.Count) / ((g.nodes.Count -
1) * d);
                    unsorted_closeness.Add(deg);
                }
Degree deg2 = newDegree();
                deg2.id = n.id;
Dictionary<string, int> dist2 = GetGeod(g, n, false);
```

```
int d2 = 0;
foreach (var geo in dist2)
{
                    d2 += (int)geo.Value;
                }
if (d2 != 0)
{
                    deg2.deg = (float)(dist2.Keys.Count * dist2.Keys.Count) / ((g.nodes.Count
- 1) * d2);
unsorted_proximity.Add(deg2);
                }
            }
List<Degree> clos_cent = SortDegree(unsorted_closeness);
List<Degree> prox_prest = SortDegree(unsorted_proximity);
            result.clos_cent = clos_cent;
            result.prox_prest = prox_prest;
return result;
        }

privateList<Degree> SortDegree(List<Degree> input)
        {
List<Degree> result = newList<Degree>();
var sort =
from d in input
orderby d.deg descending
selectnew { d.id, d.deg };
var res = sort.ToList();
foreach (var v in res)
{
Degree d = newDegree();
                d.id = v.id;
                d.deg = v.deg;
result.Add(d);
            }
return result;
        }

privateDictionary<string, int> GetGeod(Graph g, Node s, bool type)
        {
Dictionary<string, int> result = newDictionary<string, int>();
int dist = 0;
List<Adjacency> nodes = type ? s.Out_Adjacency : s.In_Adjacency;
bool loop = true;
while (loop == true)
            {
if (dist == 0)
                {
dist += 1;
foreach (Adjacency n in nodes)
                    {
result.Add(n.id, dist);
                    }
loop = true;
                }
else
                {
List<string> last =
                        (from r in result
where r.Value == dist
select r.Key).ToList();
if (last.Count != 0)
                    {
```

```
foreach (string l in last)
                        {
var check = g.nodes.Find(x => x.id == l);
if (check != null)
                            {
List<Adjacency> nodes2 = type ? check.Out_Adjacency : check.In_Adjacency;
foreach (Adjacency o in nodes2)
                                {
if (!result.ContainsKey(o.id))
                                    {
result.Add(o.id, dist + 1);
                                    }
                                }
                            }
                        }
dist += 1;
loop = true;
                    }
else
loop = false;
                }
            }
return result;
        }
    }
}


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Entities;
using System.Windows.Forms.DataVisualization.Charting;
using System.Threading;
using DAL;

namespace GUI
{
publicpartialclassGraph_GUI : Form
    {
privateOutput data = newOutput();
privateOutput start_data = newOutput();
privatestring first_node = string.Empty;
privateDataPoint first_dp = newDataPoint();
privateDictionary<DataPoint, string> points_dict = newDictionary<DataPoint, string>();
float minx = 0;
float maxx = 0;
float miny = 0;
float maxy = 0;
privateDataPoint selected_dp;
privateList<GraphNode> Last_iter_gn = newList<GraphNode>();

public Graph_GUI()
        {
InitializeComponent();
        }
```

```
privatevoid Graph_GUI_Load(object sender, EventArgs e)
        {
DBManager db = newDBManager();
List<string> link = db.GetA();
            cust_comboBox.DataSource = link;
            loading_progressBar.Minimum = 0;
            loading_progressBar.Maximum = 5;
            loading_progressBar.Value = 0;
List<string> colors = newList<string>();
colors.Add("Blue");
colors.Add("Green");
colors.Add("Black");
            color_comboBox.DataSource = colors;
            Yes_Button.Checked = true;
            magnet_no.Checked = true;
            Cent_Deg_Button.Checked = true;
            Bubble_Button.Checked = true;
            stable_no.Checked = true;
            Select_Button.Checked = true;
        }

privatevoid ForceDirectedDraw(Output graph, int detail, int iter)
        {
List<GraphNode> temp = RandomPositions(graph);
List<GraphNode> nodes = newList<GraphNode>();
if (No_Button.Checked)
nodes = RemoveBasicNode(temp);
if (Yes_Button.Checked)
nodes = temp;


for (int i = 1; i <= iter; i++)
            {
if (i == 1)
                {
                    progressBar2.Minimum = 1;
                    progressBar2.Maximum = iter * nodes.Count;

Series series = InitiatingGraph(nodes);

AssignDataPoints(nodes, series, i, iter);

chart1.Series.Add(series);
if (stable_no.Checked)
                    {
chart1.ChartAreas[0].AxisX.Minimum = (double)1.05 * minx;
chart1.ChartAreas[0].AxisX.Maximum = (double)1.05 * maxx;
chart1.ChartAreas[0].AxisY.Minimum = (double)1.05 * miny;
chart1.ChartAreas[0].AxisY.Maximum = (double)1.05 * maxy;
                    }
elseif (stable_yes.Checked)
                    {
chart1.ChartAreas[0].AxisX.Minimum = -1 * nodes.Count;
chart1.ChartAreas[0].AxisX.Maximum = 1 * nodes.Count;
chart1.ChartAreas[0].AxisY.Minimum = -1 * nodes.Count;
chart1.ChartAreas[0].AxisY.Maximum = 1 * nodes.Count;
                    }


AssignAnnotations(nodes);

Application.DoEvents();
```

```
Thread.Sleep(233);

                }
else
                {
chart1.Series.Clear();
chart1.Annotations.Clear();
ApplyForces(nodes, graph.total_dur, i);

Series series = InitiatingGraph(nodes);

AssignDataPoints(nodes, series, i, iter);

chart1.Series.Add(series);
if (stable_no.Checked)
                    {
chart1.ChartAreas[0].AxisX.Minimum = (double)1.05 * minx;
chart1.ChartAreas[0].AxisX.Maximum = (double)1.05 * maxx;
chart1.ChartAreas[0].AxisY.Minimum = (double)1.05 * miny;
chart1.ChartAreas[0].AxisY.Maximum = (double)1.05 * maxy;
                    }
elseif (stable_yes.Checked)
                    {
chart1.ChartAreas[0].AxisX.Minimum = -1.5 * nodes.Count;
chart1.ChartAreas[0].AxisX.Maximum = 1.5 * nodes.Count;
chart1.ChartAreas[0].AxisY.Minimum = -1.5 * nodes.Count;
chart1.ChartAreas[0].AxisY.Maximum = 1.5 * nodes.Count;
                    }

AssignAnnotations(nodes);

if (i == 1 | i == 2 | i % 5 == 0)
                    {
Application.DoEvents();
Thread.Sleep(233);
                    }
                }
            }
        }

privateSeries InitiatingGraph(List<GraphNode> nodes)
        {
progressBar2.Increment(nodes.Count);
Series series = newSeries();
            series.ChartType = SeriesChartType.Bubble;
            series.MarkerStyle = MarkerStyle.Circle;
DataPoint hidden = newDataPoint();
Degree deg = newDegree();
if (Cent_Deg_Button.Checked)
deg = data.cent_deg[0];
elseif (Clos_Cent_Button.Checked)
deg = data.clos_cent[0];
elseif (Prest_Deg_Button.Checked)
deg = data.prest_deg[0];
elseif (clust_coeff_Button.Checked)
deg = data.clust_coef[0];
elseif (Prox_Prest_Button.Checked)
deg = data.prox_prest[0];
int size = Convert.ToInt16(deg.deg * 120);
hidden.SetValueXY(0, 0, size);
            hidden.Color = Color.Transparent;
series.Points.Add(hidden);
```

```
switch (color_comboBox.Text)
            {
case"Blue":
                    series.Color = Color.Blue;
break;
case"Green":
                    series.Color = Color.Green;
break;
case"Black":
                    series.Color = Color.Black;
break;
default:
break;
            }
return series;
        }

privatevoid AssignAnnotations(List<GraphNode> nodes)
        {
foreach (var p1 in chart1.Series[0].Points)
            {
if (p1.XValue != 0 && p1.YValues[0] != 0)
                {
float find_x = (float)p1.XValue;
GraphNode n = nodes.Find(x => x.X == find_x);
foreach (Adjacency adj in n.Out_Adjacency)
                    {
GraphNode n2 = nodes.Find(x => x.id == adj.id);
if (Yes_Button.Checked)
                        {
DataPoint p2 = chart1.Series[0].Points.FindByValue(n2.X, "X");
LineAnnotation ann = newLineAnnotation();
ann.SetAnchor(p1, p2);
                            ann.EndCap = LineAnchorCapStyle.Arrow;
                            ann.LineWidth = SetAnnSize(adj.dur, data.total_dur);
if (adj.dur < data.nodes.Count * trackBar2.Value)
                                ann.LineDashStyle = ChartDashStyle.Dot;
chart1.Annotations.Add(ann);
                        }
elseif (n2 != null)
                        {
DataPoint p2 = chart1.Series[0].Points.FindByValue(n2.X, "X");
LineAnnotation ann = newLineAnnotation();
ann.SetAnchor(p1, p2);
                            ann.EndCap = LineAnchorCapStyle.Arrow;
                            ann.LineWidth = SetAnnSize(adj.dur, data.total_dur);
if (adj.dur < data.nodes.Count * trackBar2.Value)
                                ann.LineDashStyle = ChartDashStyle.Dot;
chart1.Annotations.Add(ann);
                        }
                    }
                }
            }
        }

privateDataPoint FixAnn(DataPoint start, DataPoint end)
        {
DataPoint result = newDataPoint();
if (end.XValue - start.XValue > 0)
                result.XValue += 2;
else
```

```
                result.XValue -= 2;
if (end.YValues[0] - start.YValues[0] > 0)
result.YValues[0] += 2;
else
result.YValues[0] -= 2;
return result;
        }

privatevoid AssignDataPoints(List<GraphNode> nodes, Series series, int i, int iter)
        {
            Last_iter_gn.Clear();
            points_dict.Clear();
foreach (GraphNode n in nodes)
            {
DataPoint p1 = newDataPoint();

if (n.id == first_node)
                    p1.Color = Color.Red;

if (stable_yes.Checked && n.id == first_node)
                {
                    n.X = -1;
                    n.Y = -1;
                }

p1.SetValueXY(n.X, n.Y, n.size);
                p1.BorderColor = Color.Black;
series.Points.Add(p1);
if (n.X < minx)
minx = n.X;
if (n.X > maxx)
maxx = n.X;
if (n.Y < miny)
miny = n.Y;
if (n.Y > maxy)
maxy = n.Y;

if (i == iter)
                {
                    Last_iter_gn.Add(n);
                    points_dict.Add(p1, n.id);
if (first_node == n.id)
                        first_dp = p1;
                }
            }
        }

privatevoid ApplyForces(List<GraphNode> nodes, int total_dur, int iter)
        {
float k = (float)Math.Sqrt((maxx - minx) * (maxy - miny) / nodes.Count());
foreach (GraphNode n in nodes)
            {
                n.force_x = 0;
                n.force_y = 0;
            }

foreach (GraphNode n1 in nodes)
            {

if (stable_yes.Checked && n1.id == first_node)
                {
                    n1.X = data.nodes.Count / 2;
```

```
                    n1.Y = data.nodes.Count / 2;
                }
foreach (GraphNode n2 in nodes)
                {
if (n1 != n2)
                    {
float d = EuclideanDistance(n1.X, n1.Y, n2.X, n2.Y);
Adjacency adj1 = null;
if (Cent_Deg_Button.Checked | Clos_Cent_Button.Checked | clust_coeff_Button.Checked)
                            adj1 = n1.Out_Adjacency.Find(x => x.id == n2.id);
elseif (Prest_Deg_Button.Checked)
                            adj1 = n1.In_Adjacency.Find(x => x.id == n2.id);
if (adj1 != null)
                        {
float len = adj1.dur;
Adjacency adj2 = null;
if (Cent_Deg_Button.Checked | Clos_Cent_Button.Checked | clust_coeff_Button.Checked)
                                adj2 = n1.Out_Adjacency.Find(x => x.id == n1.id);
elseif (Prest_Deg_Button.Checked)
                                adj2 = n1.In_Adjacency.Find(x => x.id == n1.id);
if (adj2 != null)
                            {
len += adj2.dur;
                            }
                            n1.force_x -= (float)(((n1.X - n2.X) / d) * AttrForce(d, (maxx -
minx) / (n1.size + n2.size) + k / len));
                            n1.force_y -= (float)(((n1.Y - n2.Y) / d) * AttrForce(d, (maxy -
miny) / (n1.size + n2.size) + k / len));
                            n2.force_x += (float)(((n1.X - n2.X) / d) * AttrForce(d, (maxx -
minx) / (n1.size + n2.size) + k / len));
                            n2.force_y += (float)(((n1.Y - n2.Y) / d) * AttrForce(d, (maxy -
miny) / (n1.size + n2.size) + k / len));
                        }
else
                        {
                            n1.force_x += (float)(((n1.X - n2.X) / d) * RepulForce(d, k));
                            n1.force_y += (float)(((n1.Y - n2.Y) / d) * RepulForce(d, k));
                            n2.force_x -= (float)(((n1.X - n2.X) / d) * RepulForce(d, k));
                            n2.force_y -= (float)(((n1.Y - n2.Y) / d) * RepulForce(d, k));
}

if (magnet_yes.Checked)
                        {
if (n1.Out_Adjacency.Count == 0 && iter % 3 == 0)
                            {
                                n1.force_x += 1;
                            }
                        }
if (stable_yes.Checked && n1.id == first_node | n2.id == first_node)
                        {
                            n1.force_x = n1.force_y = n2.force_x = n2.force_y = 0;
                        }
                    }
                }
if (float.IsNaN(n1.force_x) || float.IsNaN(n1.force_y))
thrownewArithmeticException();
else
                {
                    n1.X += (float)0.10 * n1.force_x;
                    n1.Y += (float)0.10 * n1.force_y;
}
            }
```

```
        }

privatefloat EuclideanDistance(float x1, float y1, float x2, float y2)
{
float result = 0;
result = (float)(Math.Sqrt(Math.Pow((x1 - x2), 2) + Math.Pow((y1 - y2), 2)));
return result;
        }

privatefloat AttrForce(float dist, float k)
        {
float result = (float)(Math.Log(dist / k));
return result;
        }

privatefloat RepulForce(float dist, float k)
        {
float result = (float)(k / Math.Pow(dist, 2));
//float result = (float)(10*k / dist);
return result;
        }

privateList<GraphNode> RemoveBasicNode(List<GraphNode> nodes)
{
List<GraphNode> result = newList<GraphNode>();
GraphNode basic = nodes.Find(x => x.id == first_node);
foreach (GraphNode g in nodes)
{
if (g != basic)
result.Add(g);
}
return result;
        }

privateList<GraphNode> RandomPositions(Output graph)
        {
List<GraphNode> result = Conversion(graph.nodes);
int k = 0;
            k = (int)graph.nodes.Count;
List<Point> points = newList<Point>();
foreach (GraphNode gn in result)
{
Random r1 = newRandom();
Point p = newPoint();
p.X = r1.Next(0, k);
                p.Y = r1.Next(0, k);
Point np = PointCheck(points, p, k);
gn.X = np.X;
                gn.Y = np.Y;
points.Add(np);

            }
return result;
        }

privatePoint PointCheck(List<Point> list, Point p, int k)
        {
Point temp = p;
while (list.Contains(temp))
{
Random r = newRandom();
```

```
                temp.X = r.Next(0, k);
                temp.Y = r.Next(0, k);
            }
return temp;
        }

privateint SetAnnSize(int dur, int total)
        {
int res = 0;
for (int i = 1; i <= 9; i++)
            {
if (i == 1)
if (dur < total / 10)
res = 1;
if (dur > total * (i) / 10 && dur <= total * (i + 1) / 10)
res = i;
            }
return res;
        }

publicList<GraphNode> Conversion(List<Node> nodes)
        {
List<GraphNode> result = newList<GraphNode>();
foreach (Node nod in nodes)
            {

GraphNode gn = newGraphNode();
                gn.id = nod.id;
                gn.Out_Adjacency = nod.Out_Adjacency;
                gn.In_Adjacency = nod.In_Adjacency;
                gn.level = nod.level;
if (Bubble_Button.Checked)
                {
Degree deg = newDegree();
if (Cent_Deg_Button.Checked)
deg = data.cent_deg.Find(x => x.id == nod.id);
elseif (Clos_Cent_Button.Checked)
deg = data.clos_cent.Find(x => x.id == nod.id);
elseif (Prest_Deg_Button.Checked)
deg = data.prest_deg.Find(x => x.id == nod.id);
elseif (clust_coeff_Button.Checked)
deg = data.clust_coef.Find(x => x.id == nod.id);
elseif (Prox_Prest_Button.Checked)
deg = data.prox_prest.Find(x => x.id == nod.id);
if (deg != null)
                        gn.size = Convert.ToInt16(deg.deg * 100);
else
                        gn.size = 1;
                }
else
                    gn.size = 10;
result.Add(gn);
            }
return result;
        }

privatevoid clearbutton_click(object sender, EventArgs e)
        {
chart1.Series.Clear();
chart1.Annotations.Clear();
            progressBar2.Value = 1;
            dataGridView1.DataSource = null;
```

```
            points_dict.Clear();
maxx = 0;
maxy = 0;
minx = 0;
miny = 0;
        }

privatevoid drawbutton_click(object sender, EventArgs e)
        {
            loading_progressBar.Value = 0;
int detail = (int)trackBar2.Value;
int iter = trackBar4.Value;
ForceDirectedDraw(data, detail, iter);
        }

privatevoid create_button_Click(object sender, EventArgs e)
        {
            points_dict.Clear();
            total_nodes_label.Text = "Nodes: ";
            total_dur_label.Text = "Total Duration: ";
vertex_Label.Text = "Vertices: ";
            level_label.Text = "Level: ";
            first_node_label.Text = "First Node: ";
Application.DoEvents();
            loading_progressBar.Value = 0;
            loading_progressBar.Increment(2);
string a = (string)cust_comboBox.Text;
byte lvl = (byte)level_trackBar.Value;
Metrics ms = newMetrics();
Graph g = ms.GetNetwork(a, lvl);
            data.nodes = g.nodes;
            data.total_dur = g.total_dur;
            loading_progressBar.Increment(1);
Output deg = ms.Cent_Prest_Degrees(g);
            data.cent_deg = deg.cent_deg;
            data.prest_deg = deg.prest_deg;
            loading_progressBar.Increment(1);
Output deg2 = ms.Clos_Cent_Prox_Prest(g);
            data.clos_cent = deg2.clos_cent;
            loading_progressBar.Increment(1);
            data.clust_coef = ms.ClusteringCoefficiant(g);
            loading_progressBar.Increment(1);
            data.prox_prest = deg2.prox_prest;
            loading_progressBar.Increment(1);
            total_nodes_label.Text += Convert.ToString(g.nodes.Count());
            total_dur_label.Text += Convert.ToString(g.total_dur);
            level_label.Text += Convert.ToString(level_trackBar.Value);
            first_node_label.Text += a;
            first_node = a;
int total_vertices = 0;
foreach (Node n in g.nodes)
            {
                total_vertices += n.Out_Adjacency.Count();
            }
            vertex_Label.Text += Convert.ToString(total_vertices);
chart1.Series.Clear();
chart1.Annotations.Clear();
        }

privatevoid chart1_MouseDown(object sender, MouseEventArgs e)
        {
if (Move_Button.Checked)
```

```
            {
HitTestResult result = chart1.HitTest(e.X, e.Y);

if (result.ChartElementType == ChartElementType.DataPoint)
                {
DataPoint point = chart1.Series[0].Points[result.PointIndex];
                    selected_dp = point;
                }
            }
if (zoom_Button.Checked)
            {
chart1.ChartAreas[0].CursorX.IsUserEnabled = true;
chart1.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
chart1.ChartAreas[0].AxisX.ScaleView.MinSize = 0;
chart1.ChartAreas[0].CursorX.Interval = 1;
chart1.ChartAreas[0].CursorY.IsUserEnabled = true;
chart1.ChartAreas[0].CursorY.IsUserSelectionEnabled = true;
chart1.ChartAreas[0].AxisY.ScaleView.MinSize = 0;
chart1.ChartAreas[0].CursorY.Interval = 1;
chart1.ChartAreas[0].CursorX.LineColor = Color.Transparent;
chart1.ChartAreas[0].CursorY.LineColor = Color.Transparent;
}
else
            {
chart1.ChartAreas[0].CursorX.LineColor = Color.Transparent;
chart1.ChartAreas[0].CursorY.LineColor = Color.Transparent;
chart1.ChartAreas[0].CursorX.IsUserEnabled = false;
chart1.ChartAreas[0].CursorX.IsUserSelectionEnabled = false;
chart1.ChartAreas[0].CursorY.IsUserEnabled = false;
chart1.ChartAreas[0].CursorY.IsUserSelectionEnabled = false;
            }

        }

privatevoid chart1_MouseUp(object sender, MouseEventArgs e)
{
if (Move_Button.Checked)
            {

if (selected_dp != null&& points_dict.ContainsKey(selected_dp))
                {
string id = points_dict[selected_dp];
                    selected_dp.XValue =
chart1.ChartAreas[0].AxisX.PixelPositionToValue(e.X);
                    selected_dp.YValues[0] =
chart1.ChartAreas[0].AxisY.PixelPositionToValue(e.Y);
GraphNode gn1 = Last_iter_gn.Find(x => x.id == id);
                    gn1.X = (float)selected_dp.XValue;
                    gn1.Y = (float)selected_dp.YValues[0];


List<GraphNode> nodes = newList<GraphNode>();
foreach (GraphNode gn in Last_iter_gn)
                    {
nodes.Add(gn);
                    }
chart1.Series.Clear();
chart1.Annotations.Clear();
Series series = InitiatingGraph(nodes);
ApplyForces(nodes, data.total_dur, 1);
AssignDataPoints(nodes, series, 1, 1);
chart1.Series.Add(series);
```

```
chart1.ChartAreas[0].AxisX.Minimum = (double)1.05 * minx;
chart1.ChartAreas[0].AxisX.Maximum = (double)1.05 * maxx;
chart1.ChartAreas[0].AxisY.Minimum = (double)1.05 * miny;
chart1.ChartAreas[0].AxisY.Maximum = (double)1.05 * maxy;
AssignAnnotations(nodes);
Application.DoEvents();
Thread.Sleep(233);
//}
                }
            }
        }

privatevoid chart1_MouseClick(object sender, MouseEventArgs e)
        {
if (Select_Button.Checked)
            {
chart1.ChartAreas[0].CursorX.LineColor = Color.Transparent;
chart1.ChartAreas[0].CursorY.LineColor = Color.Transparent;
chart1.ChartAreas[0].CursorX.IsUserEnabled = false;
chart1.ChartAreas[0].CursorX.IsUserSelectionEnabled = false;
chart1.ChartAreas[0].CursorY.IsUserEnabled = false;
chart1.ChartAreas[0].CursorY.IsUserSelectionEnabled = false;
                dataGridView1.DataSource = null;
// Call HitTest
HitTestResult result = chart1.HitTest(e.X, e.Y);

if (result.ChartElementType == ChartElementType.DataPoint)
                {
DataPoint point = chart1.Series[0].Points[result.PointIndex];
string id = points_dict[point];
                    point.Label = id;
                    point.LabelForeColor = Color.Red;
DataTable d1 = newDataTable();
d1.Columns.Add("Property");
d1.Columns.Add("Value");
d1.Rows.Add("Node_id", id);
Degree cent_deg = data.cent_deg.Find(x => x.id == id);
if (cent_deg != null)
d1.Rows.Add("Centrality Degree", cent_deg.deg);
else
d1.Rows.Add("Centrality Degree", 0);
Degree clos_cent = data.clos_cent.Find(x => x.id == id);
if (clos_cent != null)
d1.Rows.Add("Closeness Centrallity", clos_cent.deg);
else
d1.Rows.Add("Closeness Centrallity", 0);
Degree clust = data.clust_coef.Find(x => x.id == id);
if (clust != null)
d1.Rows.Add("Clustering Coefficient", clust.deg);
else
d1.Rows.Add("Clustering Coefficient", 0);
Degree pre = data.prest_deg.Find(x => x.id == id);
if (pre != null)
d1.Rows.Add("Prestige Degree", pre.deg);
else
d1.Rows.Add("Prestige Degree", 0);
Degree prox = data.prox_prest.Find(x => x.id == id);
if (prox != null)
d1.Rows.Add("Proximity Prestige", prox.deg);
else
d1.Rows.Add("Proximity Prestige", 0);
```

```
Node n = data.nodes.Find(x => x.id == id);
int in_dur = 0;
int out_dur = 0;

foreach (Adjacency ad in n.Out_Adjacency)
                    {
                        out_dur += ad.dur;
if (Yes_Button.Checked)
                        {
if (Cent_Deg_Button.Checked | Clos_Cent_Button.Checked | clust_coeff_Button.Checked)
                            {
DataPoint p = points_dict.First(x => x.Value == ad.id).Key;
                                p.Color = Color.Yellow;
                                p.Label = ad.id;
                            }
                        }
elseif (ad.id != first_node)
                        {
if (Cent_Deg_Button.Checked | Clos_Cent_Button.Checked | clust_coeff_Button.Checked)
                            {
DataPoint p = points_dict.First(x => x.Value == ad.id).Key;
                                p.Color = Color.Yellow;
                                p.Label = ad.id;
                            }
                        }

                    }

foreach (Adjacency ad in n.In_Adjacency)
                    {
                        in_dur += ad.dur;
if (Yes_Button.Checked)
                        {
if (Prest_Deg_Button.Checked | Prox_Prest_Button.Checked)
                            {
DataPoint p = points_dict.First(x => x.Value == ad.id).Key;
                                p.Color = Color.Yellow;
                                p.Label = ad.id;
                            }
                        }
elseif (ad.id != first_node)
                        {
if (Prest_Deg_Button.Checked)
                            {
DataPoint p = points_dict.First(x => x.Value == ad.id).Key;
                                p.Color = Color.Yellow;
                                p.Label = ad.id;
                            }
                        }
                    }

d1.Rows.Add("Outgoing Duration", out_dur);
d1.Rows.Add("Ingoing Duration", in_dur);
                    dataGridView1.DataSource = d1;
                    dataGridView1.ColumnHeadersHeight = 110;
                }
else
                {
// Set default cursor
this.Cursor = Cursors.Default;
foreach (DataPoint p in chart1.Series[0].Points)
                    {
```

```
if (p.XValue != 0 && p.YValues[0] != 0)
                        {
switch (color_comboBox.Text)
                            {
case"Blue":
                                    p.Color = Color.Blue;
break;
case"Green":
                                    p.Color = Color.Green;
break;
case"Black":
                                    p.Color = Color.Black;
break;
default:
break;
                            }
                        }
                        p.Label = null;
                        p.LabelForeColor = Color.Black;
                    }
                }
if (first_dp.Color != Color.Yellow)
                    first_dp.Color = Color.Red;
Application.DoEvents();
            }

        }

    }
}
```