Ατομική Διπλωματική Εργασία

ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ ΔΟΜΙΚΩΝ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΧΑΡΑΚΤΗΡΙΣΤΙΚΩΝ ΘΕΩΡΙΩΝ ΕΠΙΧΕΙΡΗΜΑΤΟΛΟΓΙΑΣ

Στέφανος Πάντζιαρος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Στέφανος Πάντζιαρος

Επιβλέπων Καθηγητής

Γιάννης Δημόπουλος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Ιούνιος 2021

Ευχαριστίες

Για την επιτυχή ολοκλήρωση της παρούσας διπλωματικής εργασίας θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, Δρ. Γιάννη Δημόπουλο, καθηγητή του Τμήματος Πληροφορικής για την ευκαιρία που μου έδωσε να ασχοληθώ με το συγκεκριμένο θέμα μελετώντας σημαντικά θέματα επιχειρηματολογίας. Επίσης θα ήθελα να τον ευχαριστήσω για την όλη καθοδήγηση κατά την διάρκεια εκπόνησης της εργασίας μου.

Στη συνέχεια θα ήθελα να ευχαριστήσω το τμήμα Πληροφορικής, όλο το διδακτικό προσωπικό που μου έδωσαν τα κατάλληλα εφόδια για να καταφέρω να ολοκληρώσω με επιτυχία το πρόγραμμα σπουδών του τμήματος. Με αυτά που αποκόμισα μέσα από τις ώρες διδασκαλίας και των εμπειριών από το Πανεπιστήμιο Κύπρου είμαι πλέον σε θέση να είμαι ανταγωνιστικός σε όλες τις δυσκολίες που θα αντιμετωπίσω στην μετέπειτα πορεία που στο τομές της Πληροφορικής.

Περίληψη

Στις μέρες μας η επιχειρηματολογία μελετάτε από τον κλάδο της πληροφορικής όπως επίσης και από πολλούς άλλους. Η επιχειρηματολογία αποτελείται από επιχειρήματα και τις σχέσεις που έχουν τα επιχειρήματα μεταξύ τους. Τα α επιχειρήματα αυτά χρησιμοποιούνται για να υπερασπιστούν κάποια άποψη που έχει αυτός που τα χρησιμοποιεί.

Η διπλωματική αυτή έχει ως στόχο την μελέτη διαφόρων μεγάλων και σύνθετων θεωριών επιχειρηματολογίας. Για να μελετηθούν οι θεωρίες αυτές θα ασχοληθούμε με την δημιουργία των θεωριών αυτών και την επεξεργασία τους ώστε να δημιουργηθεί στο τέλος η θεωρία που θέλουμε να μελετήσουμε.

Τα επιχειρήματα έχουν κάποιες σχέσεις μεταξύ τους, είτε συμμαχικές είτε εχθρικές. Κάποια επιχειρήματα αμύνονται είτε για τον εαυτό τους είτε για άλλα επιχειρήματα και προσπαθούν αν αποκρούσουν επιθέσεις που γίνονται από άλλα επιχειρήματα. Τα επιχειρήματα μπορεί να είναι αληθή και κάποια άλλα όχι. Τα επιχειρήματα που δεν δέχονται επίθεση από κανένα άλλο επιχείρημα είναι αληθή, αν τα επιχειρήματα δέχονται από κάποιο άλλο επιχείρημα επίθεση τότε θα πρέπει να μελετηθεί η σύγκρουση αυτή και να εξεταστεί ποιο επιχείρημα είναι αληθή και ποιο όχι σε αυτή τη σύγκρουση.

Σε αυτή την διπλωματική θα ασχοληθούμε περισσότερο με τα Ευσταθή μοντέλα. Με κάποια κριτήρια δημιουργείται ένα σύνολο από επιχειρήματα τα οποία είναι αληθή και αυτά μπορούν αν επιβιώσουν μέσα από κάθε σύγκρουση με το να επιτίθονται σε όλα τα άλλα επιχειρήματα που επιτίθονται στα επιχειρήματα που βρίσκονται μέσα στο σύνολο αυτό.

Οι θεωρίες επιχειρηματολογίας θα αναπαρασταθούν ως γράφοι, με κάθε κόμβο να παριστάνει ένα επιχείρημα και οι ακμές θα παριστάνουν τις επιθέσεις μεταξύ των επιχειρημάτων. Με ένα πρόγραμμα στην γλώσσα προγραμματισμού python θα δημιουργηθούν διάφορα συνδυασμοί τυχαίων γραφημάτων με διαφορετικά χαρακτηριστικά. Με κάποια πειράματα και με ένα πρόγραμμα συνόλου απαντήσεων θα μελετηθούν οι θεωρίες αυτές. Θα καταγράψουμε τον αριθμών των μοντέλων που εντοπίστηκαν και τον χρόνο που χρειάστηκε για να εντοπιστούν και θα εξάγουμε κάποια συμπεράσματα.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Η επιχειρηματολογία ίσως είναι από τα πιο συνηθισμένα είδη του λόγου, είναι η προσπάθεια να πείσουμε τον ακροατή ή των αναγνώστη ότι μια άποψη μας είναι σωστή. Καθημερινά οι άνθρωποι προσπαθούν να πείσουν τους άλλους από τα πιο απλά ζητήματα, όπως γιατί να φάμε αυτό το φαγητό και όχι το άλλο ή γιατί να πάμε σε αυτήν την ταινία και όχι την άλλη, ως τα πιο σύνθετα και πολύπλοκα, όπως γιατί να ακολουθήσει το κράτος αυτήν την πολιτική και όχι την άλλη.

Η επιχειρηματολογία μελετάτε και στην επιστήμη της πληροφορικής. Σε μία συζήτηση επιχειρηματολογίας υπάρχουν οι δύο πλευρές, η κάθε πλευρά παραθέτει τα δικά της επιχειρήματα. Το κάθε επιχείρημα μπορεί να επιτίθεται ή να αμύνεται σε κάποια άλλα επιχειρήματα. Η κάθε πλευρά προσπαθεί με τα δικά της επιχειρήματα να επιτεθεί στα επιχειρήματα τις αντίπαλης πλευράς.

Στην πληροφορική ένα επιχειρηματολογικό πλαίσιο μπορεί να αναπαρασταθεί και ως ένας κατευθυνόμενος γράφος. Σε ένα κατευθυνόμενο γράφο οι κορυφές είναι τα επιχειρήματα και οι ακμές είναι οι επιθέσεις που υπάρχουν μεταξύ των επιχειρημάτων. Μέσα σε αυτούς τους γράφους προσπαθούμε να βρούμε ένα σύνολο επιχειρημάτων που να είναι αποδεκτά. Αποδεκτά είναι τα επιχειρήματα που μπορούν να υπάρξουν χωρίς να τους επιτίθεται κάποιο άλλο επιχείρημα. Μέσα στο σύνολο των αποδεκτών είναι τα επιχειρήματα που δεν τους επιτίθεται κανένα άλλο επιχείρημα και τα επιχειρήματα τα οποία όλα τους τα επιχειρήματα που τους επιτίθονται δεν είναι αποδεκτά. Όταν ένα επιχείρημα επιτίθεται σε ένα άλλο επιχείρημα το οποίο αυτό με την σειρά του επιτίθεται σε ένα τρίτο επιχείρημα τότε λέμε ότι το πρώτο επιχείρημα αμύνεται για το τρίτο επιχείρημα.

Στην εργασία μου υλοποίησα ένα πρόγραμμα στη γλώσσα προγραμματισμού python το οποίο δημιουργεί θεωρίες επιχειρηματολογίας. Αυτές οι θεωρίες εμφανίζονται στην μορφή ενός γράφου, με κορυφές τα επιχειρήματα και ακμές τις επιθέσεις. Το πρόγραμμα χρησιμοποιεί την βιβλιοθήκη python-igraph η οποία προσφέρει την δυνατότητα στο

χρήστη της να δημιουργεί γράφους να τους επεξεργάζεται και να τους βλέπει. Υπάρχουν υλοποιημένες μέθοδοι οι οποίες δημιουργούν τυχαία γραφήματα βασισμένα σε κάποιες θεωρίες. Με αυτές τις μεθόδους δίνω την δυνατότητα στον χρήστη να δημιουργήσει τυχαία γραφήματα μέσα από μία λίστα 9 ειδών γραφημάτων.

Στο πρόγραμμα μου ο γράφος που θα δημιουργηθεί είναι ένας φωλιασμένος γράφος στον οποίο ο υπεργράφος μπορεί να είναι ένα είδος και ο υπογράφος να είναι άλλο είδος γράφου. Ο χρήστης επιλέγει πιο είδος θέλει να είναι ο κάθε γράφος και στην συνέχεια μέσα από αρχεία εισόδου δίνει τις παραμέτρους για τους γράφους που θα δημιουργηθούν. Ο χρήστης επιλέγει επίσης με πόσες ακμές θα είναι ενωμένοι οι υπογράφου του τελικού γράφου και επίσης επιλέγει το ποσοστό των μη-κατευθυνόμενων ακμών που θα υπάρχουν στον γράφο.

Στη συνέχεια ο γράφος δημιουργείται και αυτός ο γράφος μεταφράζεται ως μία θεωρία επιχειρηματολογίας όπου κάθε κορυφή είναι ένα επιχείρημα και μία ακμή μία επίθεση από ένα επιχείρημα σε ένα άλλο. Αυτή η θεωρία αποθηκεύεται σε ένα αρχείο και αυτό το αρχείο περνά σαν είσοδο σε ένα πρόγραμμα συνόλου απαντήσεων το οποίο βρίσκει μέσα από αυτή την θεωρία ευσταθή μοντέλα. Τα ευσταθή μοντέλα είναι κάποια έννοια της σημασιολογίας της επιχειρηματικότητας τα οποία θα εξηγηθούν στην συνέχεια.

Τέλος, μπορούμε βάση των αποτελεσμάτων του προγράμματος συνόλου απαντήσεων να εντοπίσουμε κάποια μοτίβα σε κάποιες θεωρίες και να καταλήξουμε σε κάποιο συμπέρασμα. Τα συμπεράσματα μπορεί να είναι είτε βάση των πόσων μοντέλων εντόπισε το πρόγραμμα είτε ακόμη και για τον χρόνο που έκανε το πρόγραμμα να τα εντοπίσει.

Κεφάλαιο 2

Επιχειρηματολογία

2.1 Εισαγωγή στην επιχειρηματολογία

[2] Από την εποχή των αρχαίων Ελλήνων φιλοσόφων και ρητόρων, επιστήμονες της επιχειρηματολογίας έχουν ερευνήσει για απαιτήσεις οι οποίες κάνουν ένα επιχείρημα ορθό, μέσω κάποιων κατάλληλων προτύπων απόδειξης και μέσω της εξέτασης των λαθών της αιτιολογίας που κάνουμε όταν προσπαθούμε να χρησιμοποιήσουμε επιχειρήματα.

Υπάρχουν τέσσερα καθήκοντα τα οποία η επιχειρηματολογία έχει αναλάβει. Τα τέσσερα αυτά καθήκοντα λέγονται: Ταύτιση (identification) , Ανάλυση (analysis) , Αξιολόγηση (evaluation) και Εφεύρεση (invention). Το καθήκον της ταύτισης είναι να αναγνωρίζει την υπόθεση και το συμπέρασμα ενός επιχειρήματος όπως βρίσκεται σε μια συνομιλία. Ένα μέρος αυτού του καθήκοντος είναι να καθορίσει πότε ένα επιχείρημα που βρέθηκε σε ένα κείμενο ταιριάζει σε μια γνωστή μορφή επιχειρημάτων η οποία λέγεται σχήμα επιχειρηματολογίας.

Το καθήκον της ανάλυσης είναι το να βρίσκει υπονοούμενες υποθέσεις ή συμπεράσματα μέσα σε ένα επιχείρημα τα οποία πρέπει να γίνουν σαφή ως αποτέλεσμα την κατάλληλη αξιολόγηση του επιχειρήματος. Επιχειρήματα τα οποία βρίσκονται σε κείμενο φυσικής γλώσσας το οποίο προέρχεται από κάποιο διάλογο τείνουν να παραλείπουν κάποιες υποθέσεις ή υπονοούν κάποια από τα συμπεράσματά τους.

Το καθήκον της αξιολόγησης είναι να καθορίσει πότε ένα επιχείρημα είναι αδύναμο ή δυνατό βάση κάποιων γενικών κριτηρίων τα οποία μπορούν να εφαρμοστούν σε αυτά. Το καθήκον της εφεύρεσης είναι να δημιουργήσει καινούργια επιχειρήματα τα οποία μπορούν να χρησιμοποιηθούν για να αποδείξουν κάποιο συγκεκριμένο συμπέρασμα. Ιστορικά, πρόσφατες εργασίες έχουν κυρίως κατεύθυνση προς τα πρώτα τρία που αναφέρθηκαν πιο πάνω, παρ' όλα αυτά υπάρχουν κατά καιρούς προσπάθειες να αντιμετωπίσουν και το τέταρτο καθήκον.

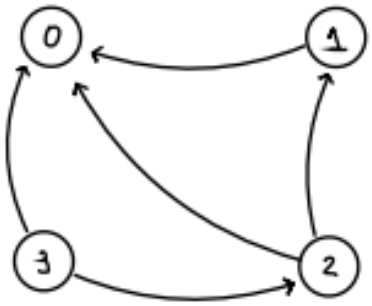
Υπάρχουν διαφορές στη βιβλιογραφία της θεωρίας της επιχειρηματολογίας για το πώς ορίζετε ένα επιχείρημα. Ένας απλός ορισμός είναι ότι ένα επιχείρημα είναι ένα σύνολο από καταστάσεις, ένα τελικό συμπέρασμα, ένα σύνολο από υποθέσεις και ένα συμπέρασμα που οδηγεί από τις υποθέσεις στο τελικό συμπέρασμα. Ένα επιχείρημα μπορεί να υποστηρίζεται από άλλα επιχειρήματα ή να δέχεται επίθεση από άλλα επιχειρήματα.

Ένας τρόπος για να επιτεθείς σε ένα επιχείρημα είναι να κάνεις μια κατάλληλη κριτική ερώτηση η οποία θα άρει αμφιβολίες για την αποδοχή του επιχειρήματος. Όταν συμβεί αυτό, το επιχείρημα προσωρινά αναιρείται μέχρι ο υποστηρικτής του επιχειρήματος μπορέσει να απαντήσει κατάλληλα στην κριτική ερώτηση. Ακόμη ένας τρόπος για να επιτεθείς σε ένα επιχείρημα είναι να άρεις αμφιβολίες σε μία από τις υποθέσεις του επιχειρήματος. Ένας τρίτος τρόπος να επιτεθείς σε ένα επιχείρημα είναι να φέρεις στο προσκήνιο ένα αντί-επιχείρημα το οποίο αντικρούει το επιχείρημα, αυτό σημαίνει ότι το συμπέρασμα του αντί-επιχειρήματος είναι η άρνηση του συμπεράσματος του επιχειρήματος.

Υπάρχουν επίσης κι άλλοι τρόποι για να επιτεθείς σε ένα επιχείρημα. Για παράδειγμα, κάποιος μπορεί να υποστηρίξει ότι οι υποθέσεις του επιχειρήματος δεν είναι σχετικές με το συμπέρασμά του ή ότι το επιχείρημα δεν σχετίζεται με το ζήτημα που υποθετικά συζητείτε. Ένας άλλος επίσης μπορεί να ισχυριστεί ότι το επιχείρημα διαπράττει λογική απάτη, όπως την απάτη της επαιτείας της ερώτησης. Ωστόσο , οι τρεις πιο πάνω τρόποι να επιτεθείς σε ένα επιχείρημα είναι σημαντικοί για να μας βοηθήσουν να καταλάβουμε την έννοια της αναίρεσης ενός επιχειρήματος. Η αναίρεση ενός επιχειρήματος είναι ένα αντίπαλο επιχείρημα το οποίο επιτίθεται στο κανονικό επιχείρημα και το νικάει.

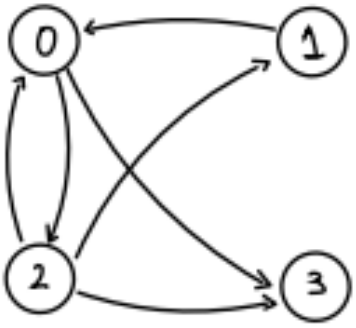
Ένας απλός τρόπος για να αναπαραστήσεις μια ακολουθία από επιχειρήματα σε ένα διάλογο είναι να θεωρήσεις τους κόμβους ενός γράφου ως τα επιχειρήματα και τις ακμές

ως τις επιθέσεις που δέχεται ένα επιχείρημα. Σε αυτού του είδους της αναπαράστασης των επιχειρημάτων, ένα επιχείρημα φαίνεται να επιτίθεται σε ένα άλλο. Για παράδειγμα στο σχήμα που ακολουθεί φαίνεται ότι το επιχείρημα 1 επιτίθεται στο επιχείρημα 0, το επιχείρημα 3 επιτίθεται στο επιχείρημα 0 και στο επιχείρημα 2 και τέλος το επιχείρημα 2 επιτίθεται στο επιχείρημα 1 και στο επιχείρημα 0.



Σχήμα 2.1 Απλή αναπαράσταση επιχειρημάτων

Πρέπει να σημειωθεί ότι τα επιχειρήματα μπορεί να επιτίθονται και στα επιχειρήματα που τους επιτίθονται. Στο σχήμα 2.2 παρατηρούμε ότι το επιχείρημα 0 επιτίθεται στο επιχείρημα 2 και το επιχείρημα 2 επιτίθεται στο επιχείρημα 0.



Σχήμα 2.2 Απλή αναπαράσταση επιχειρημάτων με επιχειρήματα που επιτίθεται το ένα στο άλλο

Στο σύστημα του Dung, οι έννοιες των επιθέσεων των επιχειρημάτων είναι απροσδιόριστες αρχές αλλά το σύστημα μπορεί να χρησιμοποιηθεί για να μοντελοποιήσει κριτήρια της αποδοχής των επιχειρημάτων. Ένα τέτοιο κριτήριο είναι ότι ένα επιχείρημα μπορεί να συμπεριληφθεί στα αποδεχτά επιχειρήματα μόνο αν κάθε

επιχείρημα που του επιτίθεται δέχεται και αυτό επίθεση από ένα επιχείρημα που βρίσκεται μέσα στο σύνολο των αποδεχτών επιχειρημάτων.

Ένα ενθύμημα(enthymeme) είναι ένα επιχείρημα με υπονοούμενες υποθέσεις και συμπεράσματα τα οποία χρειάζεται να ξεκαθαριστούν έτσι ώστε να μπορεί το επιχείρημα να κατανοηθεί ή να αξιολογηθεί κατάλληλα. Το πιο απλό παράδειγμα είναι το επιχείρημα : όλοι οι άντρες είναι θνητοί, άρα ο Σωκράτης είναι θνητός. Όπως λέγεται σε πολλές σημειώσεις λογικής, η υπόθεση ότι ο Σωκράτη είναι άντρας χρειάζεται να υποθεί έτσι ώστε το επιχείρημα να γίνει έγκυρο. Επειδή και οι δύο υποθέσεις χρειάζεται να υποστηρίζουν το συμπέρασμα επαρκώς.

Ακόμη ένα παράδειγμα για ενθύμημα είναι: Τα ζώα τα οποία βρίσκονται σε αιχμαλωσία είναι πιο ελεύθερα από το να βρίσκονται στη φύση επειδή δεν υπάρχουν φυσικά αρπακτικά για να τα σκοτώσουν. Το σαφές συμπέρασμα είναι ξεκάθαρα η πρώτη πρόταση , τα ζώα τα οποία βρίσκονται σε αιχμαλωσία είναι πιο ελεύθερα από το να βρίσκονται στη φύση. Η σαφής υπόθεση που υποστηρίζει το συμπέρασμα είναι το ότι δεν υπάρχουν φυσικά αρπακτικά για να σκοτώσουν τα ζώα που βρίσκονται σε αιχμαλωσία από το να βρίσκονται στη φύση. Στη συνέχεια έχουμε τις προτάσεις που υπονοούνται, αν τα ζώα βρίσκονται σε ένα μέρος στο οποίο δεν υπάρχουν αρπακτικά για να τα σκοτώσουν , είναι πιο ελεύθερα από το να βρίσκονται σε μέρος που υπάρχουν αρπακτικά τα οποία μπορούν να τα σκοτώσουν. Η πρώτη υπόθεση που υπονοείται είναι θέμα της κοινής λογικής. Το δεύτερο, ωστόσο , εκφράζει ένα ειδικό τρόπο που αυτός που εκφράζει αυτή την άποψη χρησιμοποιεί την λέξη ελεύθερο που φαίνεται να είναι αντίθετο της κοινής λογικής, ή δεν φαίνεται να βασίζεται σε αυτή. Φαίνεται να παρουσιάζεται η δική του ειδική θέση στη σημασία της λέξης ελευθερία.

Η υπόθεση υπάρχουν φυσικά αρπακτικά τα οποία μπορούν να σκοτώσουν τα ζώα που βρίσκονται στη φύση είναι υπονοούμενη υπόθεση που βασίζεται στη κοινή λογική. Αν τα ζώα βρίσκονται σε μέρος που δεν υπάρχουν αρπακτικά για να τα σκοτώσουν είναι πιο ελεύθερα από το να βρίσκονται σε μέρος που υπάρχουν αρπακτικά για να τα σκοτώσουν, αυτό είναι μία υπονοούμενη υπόθεση που βασίζεται στην ειδική δέσμευση αυτού που εξέφρασε αυτή την υπόθεση. Αυτά τα δύο μαζί με την υπόθεση ότι δεν υπάρχουν αρπακτικά για να σκοτώσουν τα ζώα που βρίσκονται σε αιχμαλωσία πρέπει να υποστηρίζουν και τα τρία επαρκώς το συμπέρασμα ότι τα ζώα σε αιχμαλωσία είναι πιο

ελεύθερα από τα ζώα στη φύση. Αν πάρουμε τις τρείς υποθέσεις ξεχωριστά δεν θα μπορούμε να υποστηρίξουμε το συμπέρασμα, πρέπει να πάρουμε και τις τρεις μαζί.

Υπάρχουν 6 είδη διαλόγων στη θεωρία των διαλόγων.

1. Ο Διάλογος που προσπαθείς να πείσεις τον συνομιλητή σου για κάτι
2. Ο διάλογος που κάνεις για να ερευνήσεις κάτι
3. Ο διάλογος που πρέπει να διαπραγματευτείς με κάποιον για ένα θέμα
4. Ο διάλογος που προσπαθείς να αναζητήσεις πληροφορίες
5. Ο διάλογος που γίνεται σε κάποια σύσκεψη
6. Ο διάλογος που προσπαθεί ο ένας να αποκρούσει το επιχείρημα του άλλου χωρίς να τον νοιάζει ιδιαίτερα για την αλήθεια.

Οι διάλογοι έχουν κάποιες ιδιότητες, αρχική κατάσταση, ο στόχος αυτών που συμμετέχουν στο διάλογο και τον στόχο του διαλόγου.

Η αρχική κατάσταση του πρώτου διαλόγου είναι η σύγκρουση των απόψεων αυτών που συμμετέχουν στον διάλογο. Ο στόχος τους είναι να πείσουν την αντίθετη πλευρά ότι αυτοί έχουν δίκαιο και τέλος ο στόχος αυτού του διαλόγου είναι να λύσει κάποιο θέμα που προέκυψε.

Στο δεύτερο είδος διαλόγου , η αρχική κατάσταση είναι ότι κάτι χρειάζεται μια απόδειξη. Ο στόχος αυτών που συμμετέχουν στον διάλογο είναι να βρουν και να επιβεβαιώσουν κάποια στοιχεία. Ο στόχος του διαλόγου αυτού είναι να αποδείξει μίαν υπόθεση που επέλεξαν οι συμμετέχοντες σε αυτό το διάλογο.

Στο τρίτο διάλογο η αρχική κατάσταση είναι μια κατάσταση στην οποία υπάρχει σύγκρουση ενδιαφερόντων. Ο στόχος των συμμετεχόντων είναι να κατακτήσουν αυτό που θέλουν πιο πολύ. Ο στόχος του διαλόγου αυτού είναι να βρεθεί μία λογική λύση που να μπορούν να την αποδεχτούν και οι δύο πλευρές.

Στο τέταρτο διάλογο η αρχική κατάσταση είναι η κατάσταση στην οποία ο ενδιαφερόμενος θέλει κάποιες πληροφορίες. Ο στόχος των συμμετεχόντων στο διάλογο αυτό είναι να δώσουν ή να αποκτήσουν πληροφορίες ανάλογα με την πλευρά που βρίσκονται. Ο στόχος γενικά του διαλόγου αυτού είναι η ανταλλαγή πληροφοριών.

Στο πέμπτο διάλογο η αρχική κατάσταση είναι η κατάσταση στην οποία υπάρχει ένα δίλημμα ή πρέπει οι συμμετέχοντες να πάρουν μια απόφαση. Ο στόχος των συμμετεχόντων σε αυτού του είδους διάλογο είναι να συνεργαστούν έτσι ώστε να

φτάσουν μαζί σε ένα κοινό στόχο αποφασίζοντας μαζί τις δράσεις που θα κάνουν. Ο στόχος γενικά του διαλόγου αυτού είναι να αποφασιστεί το καλύτερο διαθέσιμο σύνολο από δράσεις που θα πρέπει να κάνουν οι συμμετέχοντες.

Τέλος, η αρχική κατάσταση του έκτου διαλόγου είναι η κατάσταση στην οποία υπάρχει προσωπική σύγκρουση. Ο στόχος των συμμετεχόντων σε αυτό τον διάλογο είναι να «καταστρέψουν» τον αντίπαλο τους. Ο γενικός στόχος αυτού του διαλόγου είναι να αποκαλυφθούν βαθύτερες βάσεις αυτής της σύγκρουσης που έχουν οι συμμετέχοντες μεταξύ τους.

2.2 Σημασιολογία της επιχειρηματολογίας

Στις μέρες μας, οι περισσότερες έρευνες στο θέμα της επιχειρηματολογίας είναι βασισμένες στην αφηρημένη θεωρία επιχειρηματολογίας του Dung. Η κεντρική ιδέα στη θεωρία αυτή είναι ενός πλαισίου επιχειρηματολογίας, το οποίο μπορεί να θεωρηθεί σαν ένας κατευθυνόμενος γράφος στον οποίο τα επιχειρήματα είναι οι κόμβοι του γράφου και οι ακμές του είναι οι επιθέσεις μεταξύ των επιχειρημάτων. Με τέτοιο γράφο, μπορεί κάποιος να απαντήσει στην ερώτηση ποια είναι τα σύνολα των αποδεχτών επιχειρημάτων, το να απαντήσεις αυτή την ερώτηση έχει σχέση με το να ορίσεις μία σημασιολογία της επιχειρηματολογίας(argumentation semantics).

Αν κάποιος θέλει να χρησιμοποιήσει θεωρία επιχειρηματολογίας για τον σκοπό της επιμέλειας, μπορεί να το διασπάσει σε τρία βήματα. Πρώτα μπορεί να χρησιμοποιήσει μία βάση γνώσης για να παράξει ένα σύνολο από επιχειρήματα και να ορίσει με πιο τρόπο επιτίθεται το ένα επιχείρημα στο άλλο. Το αποτέλεσμα του πρώτου βήματος είναι ένα επιχειρηματολογικό πλαίσιο που παρουσιάζεται σαν ένα κατευθυνόμενος γράφος όπου ο κάθε κόμβος είναι το επιχείρημα και οι ακμές αναπαριστούν τις επιθέσεις μεταξύ των επιχειρημάτων. Βάση αυτού του πλαισίου, του κατευθυνόμενου γράφου, το επόμενο βήμα είναι να ορίσει ένα σύνολο από επιχειρήματα τα οποία μπορούν να θεωρηθούν αποδεχτά χρησιμοποιώντας ορισμένα κριτήρια τα οποία αντιστοιχούν σε ένα επιχειρηματολογικό semantic. Μετά από την ταυτοποίηση των συνόλων των αποδεχτών

επιχειρημάτων, θα πρέπει να ταυτοποιήσει το σύνολο των αποδεχτών συμπερασμάτων, για τα οποία υπάρχουν πολλές προσεγγίσεις.

Στο σχήμα 2.3 βλέπουμε ένα επιχειρηματολογικό πλαίσιο. Φαίνεται ότι χρειάζεται μια καλά ορισμένη μέθοδος για να αντιμετωπίσεις κάποια αυθαίρετους συνδυασμούς από επιχειρηματολογικά πλαίσια, τέτοιου είδους μέθοδοι για την ταυτοποίηση συγκρούσεων που βγαίνουν μέσα από επιχειρηματολογικά πλαίσια λέγονται επιχειρηματολογικές σημασιολογίες.

Σχήμα 2.3 Ένα απλό επιχειρηματολογικό σχέδιο

Δύο κύριες προσεγγίσεις για τον ορισμό των επιχειρηματολογικών σημασιολογιών είναι σημασιολογίες με βάση τις ετικέτες(labelled-based) και σημασιολογίες με βάση τις επεκτάσεις(extension-based). Η ιδέα στο σημασιολογία με βάση τις ετικέτες είναι να δίνεται σε κάθε επιχείρημα μια ετικέτα. Οι ετικέτες για αυτή την προσέγγιση είναι τρεις: in, out και undec. Η ετικέτα in σημαίνει ότι το επιχείρημα είναι αποδεκτό, η ετικέτα out σημαίνει ότι το επιχείρημα δεν είναι αποδεκτό και η ετικέτα undec σημαίνει ότι το επιχείρημα απείχε από τις απόψεις αν είναι αποδεκτό ή όχι και έτσι δεν μπορεί να μπει ούτε στο σύνολο των αποδεκτών ούτε στο σύνολο των μη αποδεκτών. Το κάθε επιχείρημα παίρνει ακριβώς μία ετικέτα. Στο σχήμα 2.3, κάποιος μπορεί να ξεκινήσει να αναθέτει την ετικέτα in στο επιχείρημα A, το οποίο δεν δέχεται επίθεση από κάποιο άλλο επιχείρημα. Στη συνέχεια φαίνεται ότι το επιχείρημα B από την στιγμή που δέχεται επίθεση από το επιχείρημα A πρέπει να πάρει την ετικέτα out, αφού δεν γίνεται στο υποσύνολο των αποδεκτών επιχειρημάτων να υπάρχουν επιχειρήματα τα οποία επιτίθονται σε άλλα επιχειρήματα που είναι και αυτά μέσα στο υποσύνολο των αποδεκτών. Και τέλος, μπορεί το επιχείρημα C να πάρει την ετικέτα in από την στιγμή που μέσα στο υποσύνολο των αποδεκτών επιχειρημάτων δεν υπάρχει το επιχείρημα B που του επιτίθεται. Αυτή η ανάθεση φαίνεται μια λογική ανάθεση που θα μπορούσε να σκεφτεί κάποιος. Ωστόσο, υπάρχουν κι άλλες πιθανές αναθέσεις όπως για παράδειγμα

κάποιος θα μπορούσε να αναθέσει σε όλα τα επιχειρήματα την ετικέτα in αλλά στο παράδειγμα του σχήματος 2.3 θα υπάρχει πρόβλημα με τις συγκρούσεις που υπάρχουν μεταξύ των επιχειρημάτων. Μία άλλη πιθανή ανάθεση είναι να δώσει κάποιος σε όλα τα επιχειρήματα την ετικέτα undec αλλά σε αυτή την περίπτωση το επιχείρημα Α δεν δέχεται από κάποιο άλλο επιχείρημα επίθεση τότε δεν υπάρχει λόγος να του ανατεθεί η ετικέτα undec. Μία συγκεκριμένη σημασιολογία με βάση τις ετικέτες παρέχει ένα τρόπο να επιλέγονται πάντα λογικές ετικέτες για το κάθε επιχείρημα μέσα από όλες τις πιθανές ετικέτες, σύμφωνα με κριτήρια που είναι ενσωματωμένα στον ορισμό τους.

Η ιδέα πίσω από την προσέγγιση της σημασιολογίας με βάση τις επεκτάσεις είναι η ταυτοποίηση υποσυνόλων από επιχειρήματα, τα οποία λέγονται επεκτάσεις, τα οποία μπορούν να υπάρξουν μαζί χωρίς να επιτίθεται κάποιο επιχείρημα σε κάποιο άλλο επιχείρημα που βρίσκεται και αυτό μέσα στο υποσύνολο. Στο σχήμα 2.3 κάποιος μπορεί να ξεκινήσει βάζοντας μέσα στο υποσύνολο το επιχείρημα Α αφού δεν δέχεται από κάποιο άλλο επιχείρημα επίθεση. Τότε το επιχείρημα Β δεν θα μπορεί να ανήκει και αυτό μέσα στην επέκταση που δημιουργείται επειδή δέχεται επίθεση από το επιχείρημα Α που είναι ήδη μέσα. Στη συνέχεια παρατηρείται ότι το επιχείρημα C μπορεί να μπει μέσα στην επέκταση που δημιουργείται αφού το επιχείρημα Β που του επιτίθεται δεν ανήκει στην επέκταση. Άρα τέλος δημιουργήθηκε η επέκταση {A,C}. Επίσης και σε αυτή την περίπτωση όπως και πριν με τη προσέγγιση με βάση τις ετικέτες και εδώ κάποιος μπορεί να δημιουργήσει άλλες επεκτάσεις, όπως για παράδειγμα το {A,B,C}, αλλά και πάλι αυτό το υποσύνολο δεν μπορεί να δημιουργηθεί επειδή δεν είναι συμβατό με τις συγκρούσεις που εμφανίζονται στο επιχειρηματολογικό πλαίσιο στο σχήμα 2.3. Κάποιος επίσης μπορεί να θεωρήσει το σύνολο κενό, αλλά και πάλι εδώ βλέπουμε ότι το επιχείρημα Α θα πρέπει να ανήκει σε όλα τα υποσύνολα που θα δημιουργούνται. Μία σημασιολογία με βάση τις επεκτάσεις παρέχει ένα τρόπο επιλογής λογικών συνόλων από επιχειρήματα από όλα τα πιθανά σύμφωνα με κάποια κριτήρια που είναι ενσωματωμένα στον ορισμό του.

Η έννοια της αποδοχής των επιχειρημάτων μπορεί να εξηγηθεί από μία απλή αρχή. Για κάθε επιχείρημα Α κάποιος αποδέχεται ή απορρίπτει μια εξήγηση του γιατί αυτό το επιχείρημα είναι αποδεκτό, σε σχέση με την αποδοχή ή την απόρριψη άλλων επιχειρημάτων που είναι συνδεδεμένα με το Α μέσα από την σχέση της επίθεσης. Στη προσέγγιση με βάση τις ετικέτες, το να αναθέσεις την ετικέτα in σε ένα επιχείρημα Α μπορεί να εννοηθεί με το ότι σε όλα τα επιχειρήματα τα οποία επιτίθονται στο Α τους

έχει ανατεθεί η ετικέτα out ή μπορεί να εννοηθεί ότι στο επιχείρημα Α δεν επιτίθεται κάποιο άλλο επιχείρημα, έτσι το επιχείρημα Α δεν επηρεάζεται από κάποια επίθεση. Αν το επιχείρημα Α πάρει την ετικέτα out αυτό σημαίνει ότι κάποιο επιχείρημα που επιτίθεται στο Α έχει πάρει την ετικέτα in.

Στην προσέγγιση με βάση τις επεκτάσεις τα πράγματα είναι λίγο διαφορετικά. Το να συμπεριλάβεις το επιχείρημα Α μέσα σε ένα σύνολο Ε το οποίο είναι η επέκταση σημαίνει ότι το Ε μπορεί να ακυρώσει όλους όσους επιτίθονται στο επιχείρημα Α. Με άλλα λόγια το Ε μπορεί να προστατέψει το επιχείρημα Α. Μια βασική προϋπόθεση για αυτό το σύνολο από επιχειρήματα είναι η δυνατότητα να αμύνεται για όλα τα επιχειρήματα που ανήκουν σε αυτό. Δεν επιτρέπονται συγκρούσεις μέσα σε αυτό το υποσύνολο μεταξύ επιχειρημάτων που βρίσκονται και τα δύο μέσα σε αυτό. Αυτό μας οδηγεί και στον ορισμό του συνόλου χωρίς συγκρούσεις.

Τα ολοκληρωμένα μοντέλα (complete semantics) μπορούν να θεωρηθούν και ως η τόνωση των βασικών απαιτήσεων που οδηγούν στην ιδέα της αποδοχής. Η δυνατότητα αποδοχής χρειάζεται κάποιο να μπορεί να δώσει κάποιους λόγους για την αποδοχή και την απόρριψη κάποιων επιχειρημάτων αλλά αφήνει ελεύθερο κάποιον να απέχει από οποιοδήποτε επιχείρημα, τα ολοκληρωμένα μοντέλα επίσης χρειάζονται κάποιων να απέχει μόνο αν δεν υπάρχουν καλοί λόγοι για να μην απέχει. Αν κάποιος απέχει από το να έχει μια άποψη για ένα επιχείρημα αν το αποδέχεται ή αν το απορρίπτει τότε κάποιος έχει ανεπαρκής λόγους για να αποδεκτεί ένα επιχείρημα , αυτό σημαίνει ότι δεν έχουν απορριφθεί όλα τα επιχειρήματα που επιτίθενται σε αυτό , και έχει ανεπαρκείς λόγους για να απορρίψει ένα επιχείρημα, δηλαδή κανένα από τα επιχειρήματα που του επιτίθονται δεν ανήκουν στο σύνολο των αποδεκτών επιχειρημάτων.

Στην προσέγγιση με τις ετικέτες, στον ορισμό για τα ολοκληρωμένα μοντέλα περικλείετέ μια έννοια όπου ένα επιχείρημα μπορεί να είναι νομίμως undecided.

Ένα ολοκληρωμένο μοντέλο στην προσέγγιση με τις ετικέτες είναι ετικέτα αν κάθε επιχείρημα που έχει την ετικέτα in έχει νόμιμα αυτή την ετικέτα, αν κάθε επιχείρημα που έχει την ετικέτα out έχει νόμιμα την ετικέτα out και αν κάθε επιχείρημα που έχει την ετικέτα undec έχει νόμιμα την ετικέτα undec.

Μία άλλη πρόταση για κατανόηση του ορισμού είναι:

Ας υποθέσουμε ότι Lab είναι μία ετικέτα ενός επιχειρηματολογικού πλαισίου. Η Lab είναι μία ολοκληρωμένη ετικέτα αν και μόνο αν για κάθε επιχείρημα Α που ανήκει στο πλαίσιο ισχύει ότι:

1. Α έχει την ετικέτα in αν και μόνο αν όλα τα επιχειρήματα που του επιτίθονται έχουν την ετικέτα out
2. Α έχει την ετικέτα out αν και μόνο αν τουλάχιστον ένα από τα επιχειρήματα που του επιτίθονται έχει την ετικέτα in

Παρόλα αυτά αυτή η υπόθεση δεν αναφέρει κάτι για τα επιχειρήματα που έχουν την ετικέτα undec. Αυτό βγαίνει μέσα από τα σημεία 1 και 2. Κάθε επιχείρημα το οποίο έχει την ετικέτα undec σημαίνει ότι δεν έχουν όλα τα επιχειρήματα που του επιτίθονται την ετικέτα out, διαφορετικά θα είχε την ετικέτα in βάση του $1^{ου}$ σημείου και δεν έχει έστω ένα επιχείρημα που του επιτίθεται την ετικέτα in διαφορετικά θα είχε την ετικέτα out βάση του $2^{ου}$ σημείου.

Στην δεύτερη προσέγγιση, με βάση τις επεκτάσεις(extension-based), μία ολοκληρωμένη επέκταση είναι ένα σύνολο από επιχειρήματα που δεν έχουν οποιαδήποτε σύγκρουση μεταξύ τους, δηλαδή δεν επιτίθεται κάποιο επιχείρημα που ανήκει σε αυτό το σύνολο σε άλλο επιχείρημα το οποίο ανήκει και αυτό μέσα στο σύνολο. Και σε αυτό το σύνολο ανήκουν ακριβώς τα επιχειρήματα τα οποία προστατεύονται από το σύνολο, δηλαδή υπάρχει έστω ένα επιχείρημα μέσα στο σύνολο που επιτίθεται στα επιχειρήματα που επιτίθονται στα επιχειρήματα που είναι μέσα στο σύνολο.

Στο σχήμα 2.4 που βρίσκεται πιο κάτω μπορούμε να δούμε ότι υπάρχουν 7 αποδεκτά σύνολα ετικετών, αλλά μόνο αυτά είναι ολοκληρωμένα ({A},{B},{C,D}), ({A,C},{B,D},Ø), ({A,D},{B,C},Ø). Συγκεκριμένα μπορούμε να πούμε ότι το επιχείρημα Α έχει πάντα την ετικέτα in επειδή δεν του επιτίθεται κάποιο άλλο επιχείρημα. Το επιχείρημα Β έχει την ετικέτα out επειδή το επιχείρημα που του επιτίθεται είναι το Α που αυτό έχει την ετικέτα in πάντα. Από την άλλη τα επιχειρήματα C και D μπορούν να έχουν την ετικέτα undec ή το ένα να έχει την ετικέτα in και το άλλο την ετικέτα out. Στο ίδιο σχήμα μπορεί να σημειωθεί ότι το σύνολο {A} είναι μία ολοκληρωμένη επέκταση όπως και τα σύνολα {A,C} και {A,D}.

Στο σχήμα 2.5 οι ολοκληρωμένες ετικέτες είναι (Ø,Ø,{A,B,C,D}), ({A,D}.{B,C},Ø) και ({B,D},{A,C},Ø). Οι ολοκληρωμένες επεκτάσεις είναι το κενό σύνολο , Ø , επειδή δεν

υπάρχει κάποιο επιχείρημα που να μην δέχεται από κάποιο άλλο επίθεση, επίσης ολοκληρωμένες επεκτάσεις είναι τα {A,D} και {B,D}. Μόνα τους τα επιχειρήματα A και B δεν μπορούν αν αποτελέσουν το σύνολο επειδή και τα δύο αμύνονται για το επιχείρημα D, άρα αν ένα από αυτά τα δύο επιχειρήματα ανήκουν στο αποδεκτό σύνολο πρέπει να ανήκει στο σύνολο και το D.

Στο σχήμα 2.6 οι ολοκληρωμένες ετικέτες είναι μόνο μία, η (Ø,Ø,{A,B,C}) και ολοκληρωμένη επέκταση είναι πάλι μόνο ένα, το κενό σύνολο Ø.

Αν λοιπόν θεωρηθεί ότι κάθε ολοκληρωμένη ετικέτα είναι και μία λογική θέση, κάποιος μπορεί να παρουσιάσει κάποια θέματα συγκρούσεων που εκφράζονται μέσα στο επιχειρηματολογικό πλαίσιο. Τότε πρέπει να εξεταστεί το ποια είναι η πιο γειωμένη θέση που κάποιος μπορεί να εκφράσει, η θέση που είναι πιο λίγο αμφισβητήσιμη. Η ιδέα σε αυτή τη σημασιολογία είναι να αποδέχονται μόνο τα επιχειρήματα που δεν γίνεται να μην αποδεχτούν, να απορριφθούν μόνο τα επιχειρήματα που δεν γίνετε να μην απορριφθούν και να αποφευχθούν όσα περισσότερα γίνετε. Αυτό ίσως είναι το πιο σκεπτική σημασιολογία από όλες τις σημασιολογιές που βασίζονται στα ολοκληρωμένα extensions.

Στην προσέγγιση της ετικέτας ας υποθέσουμε ότι AF = (Ar, att) είναι ένα επιχειρηματικό πλαίσιο. Η θεμελιωμένη ανάθεση ετικέτας(grounded labelling) του AF είναι όλα τα ολοκληρωμένα labellings Lab όπου in(Lab) έχει τα ελάχιστα επιχειρήματα που μπορεί να έχει.

Στην προσέγγιση των επεκτάσεων , η θεμελιωμένη ανάθεση επεκτάσεων(grounded extensions) του AF είναι τα ελάχιστα ολοκληρωμένα extensions του AF, δηλαδή αυτό που έχει τα πιο λίγα επιχειρήματα.

Στο σχήμα 2.4 η θεμελιωμένη ανάθεση ετικέτας είναι ({A},{B},{C,D}) και η θεμελιωμένη ανάθεση επεκτάσεων είναι το {A}. Στο σχήμα 2.5 η θεμελιωμένη ανάθεση ετικετών είναι το (Ø,Ø,{A,B,C,D}) και η θεμελιωμένη ανάθεση επεκτάσεων είναι το κενό σύνολο, Ø.

Όσο η σημασιολογία με την θεμελιώδη ανάθεση υιοθετεί μία πιο σκεπτική πλευρά με το να αποδέχεται τα ελάχιστα επιχειρήματα, κάποιος μπορεί να σκεφτεί μία διαφορετική άποψη. Την άποψη του να αποδέχονται όσα πιο πολλά επιχειρήματα είναι πιθανά. Για παράδειγμα, σε μία επίθεση όπου και τα δύο επιχειρήματα επιτίθονται το ένα στο άλλο,

μπορεί να λυθεί αυτή η σύγκρουση με την αποδοχή ενός από τα δύο επιχειρήματα, αλλά σίγουρα όχι και τα δύο. Η ιδέα να αποδεχτούν τα περισσότερα επιχειρήματα εκφράζεται από την προτιμώμενη σημασιολογία.

Στην προσέγγιση με τις ετικέτες υπάρχει ένας ορισμός. Ας υποθέσουμε ένα επιχειρηματολογικό πλαίσιο AF = (Ar,att). Μία προτιμώμενη ετικέτα του AF είναι μία ολοκληρωμένο ετικέτα Lab στο οποίο το in(Lab) έχει τα περισσότερα επιχειρήματα από όλες τις ολοκληρωμένες ετικέτες.

Στην προσέγγιση με τις επεκτάσεις, μία προτιμώμενη επέκταση είναι το μεγαλύτερο αποδεκτό σύνολο του AF.

Στο σχήμα 2.4 υπάρχουν δύο προτιμώμενες ετικέτες οι οποίες δίνουν λύση στις επιθέσεις που γίνονται μεταξύ των επιχειρημάτων C και D, τα ({A,C},{B,D},Ø) και ({A,D},{B,C},Ø) και οι προτιμώμενες επεκτάσεις είναι {A,C} και {A,D}. Στο σχήμα 2.5 πάλι υπάρχουν επιθέσεις που γίνονται μεταξύ δύο επιχειρημάτων, δηλαδή να επιτίθεται το ένα στο άλλο. Στο σχήμα αυτό τα επιχειρήματα που αλληλοεπιτίθονται είναι το επιχείρημα A και το επιχείρημα B. Όποιο και από τα δύο επιλεγεί να αποδεκτεί το επιχείρημα C θα απορριφθεί και το επιχείρημα D θα είναι αποδεκτό. Οι δύο προτιμώμενες ετικέτες για αυτό το σχήμα είναι ({A, D}, {B, C},Ø) και ({B, D}, {A, C}, Ø) και οι δυο προτιμώμενες επεκτάσεις είναι τα {A,D} και {B,D}

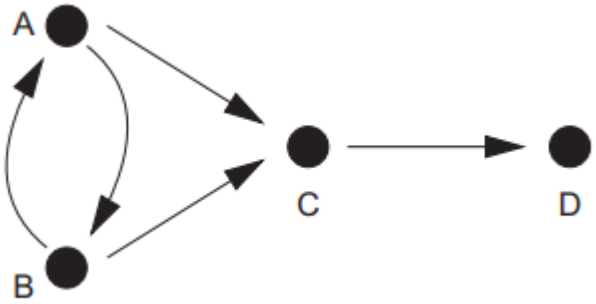
Τέλος, στο σχήμα 2.6 η μόνη αποδεκτή προτιμώμενη ετικέτα είναι (Ø,Ø,{A,B,C}) και η μόνη προτιμώμενη επέκταση είναι το κενό σύνολο, Ø.[4]

2.3 Ευσταθή μοντέλα και Σύνολα Απαντήσεων

Βάση των δύο προσεγγίσεων που ανάφερα πιο πάνω υπάρχουν διάφορα semantics. Τα ευσταθεί μοντέλα(stable semantics) είναι ένα από αυτά. Στη πρώτη προσέγγιση με βάση τις ετικέτες, Lab είναι μια ετικέτα στο επιχειρηματικό πλαίσιο AF = (Ar, att). Lab είναι μία ευσταθής ανάθεση ετικέτας του AF αν και μόνο αν undec(Lab)=0. Δηλαδή αν δεν υπάρχει ούτε ένα επιχείρημα που να μην γνωρίζουμε αν έχει την ετικέτα Lab ή όχι. Για την δεύτερη προσέγγιση, με βάση τις επεκτάσεις, ο ορισμός αλλάζει λίγο. Ας υποθέσουμε ξανά το επιχειρηματικό πλαίσιο AF = (Ar , att), μία ευσταθής ανάθεση επέκτασης(stable extensions) του AF είναι ένα υποσύνολο από επιχειρήματα που ανήκουν στο Ar τέτοια ώστε δεν επιτίθεται κανένα επιχείρημα σε άλλο επιχείρημα που είναι μέσα σε αυτό το υποσύνολο και υπάρχει έστω και μία επίθεση προς όλα τα επιχειρήματα που δεν ανήκουν μέσα σε αυτό το υποσύνολο από ένα επιχείρημα που ανήκει μέσα σε αυτό.

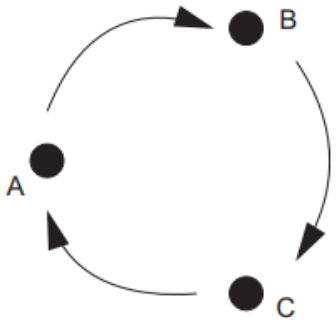
Σχήμα 2.4 Επιχειρηματολογικό πλαίσιο

Στο σχήμα 2.4 υπάρχουν δύο ευσταθείς αναθέσεις ετικέτας, τα ({A,C},{B,D},Ø) και ({A,D},{B,C},Ø). Παρομοίως υπάρχουν και δύο ευσταθείς αναθέσεις επεκτάσεις το {A,C} και το {A,D}.



Σχήμα 2.5 Επιχειρηματολογικό πλαίσιο

Στο σχήμα 2.5 οι ευσταθείς αναθέσεις ετικετών είναι τα ({A.D},{B,C},∅) και ({B,D},{A,C},∅) και οι ευσταθείς αναθέσεις επεκτάσεων είναι τα {A,D} και {B,D}.



Σχήμα 2.6 Κύκλος με τρία επιχειρήματα

Στο σχήμα 2.6 φαίνεται ότι δεν υπάρχουν πάντα ευσταθεί μοντέλα. Σε αυτό το παράδειγμα η περίπτωση του να μην υπάρχει κάποιο επιχείρημα που να μην του ανατεθεί η ετικέτα undec και η περίπτωση του να δέχονται επίθεση τα επιχειρήματα που είναι εκτός του συνόλου από αυτά που είναι εντός δεν μπορούν να πραγματοποιηθούν. Άρα σε αυτό το παράδειγμα δεν υπάρχουν ούτε ευσταθείς αναθέσεις ετικέτας ούτε ευσταθείς αναθέσεις επεκτάσεων.

Τα κύρια κομμάτια του προγραμματισμού συνόλου απαντήσεων είναι τα atoms, τα literals και οι κανόνες. Τα atoms είναι στοιχειώδεις προτάσεις που μπορεί να είναι αληθής ή ψευδής. Τα literals αποτελούνται από τα atoms α και την άρνηση του. Και οι κανόνες είναι της μορφής του σχήματος 2.7. [3]

$$a \leftarrow b_1, \ldots, b_m, not\ c_1, \ldots, not\ c_n$$

Σχήμα 2.7 Κανόνας στον προγραμματισμό συνόλου απαντήσεων

Στον κανόνα του σχήματος 2.7 το α, τα b και c είναι atoms. Ο κανόνας αυτός μπορεί να θεωρηθεί και ως την δικαιολόγηση της αποδοχής του α.

Για παράδειγμα, στον κανόνας στο σχήμα 2.8,

$$light\_on \leftarrow power\_on, not\ broken$$

Σχήμα 2.8 Κανόνας στον προγραμματισμό συνόλου απαντήσεων

μπορούμε να πούμε ότι το φως είναι ανοικτό αν καθορίσουμε ότι υπάρχει ρεύμα στον χώρο και αν καθορίσουμε ότι ο λαμπτήρας δεν έχει σπάσει για οποιοδήποτε λόγο.

Το κομμάτι αριστερά από το τόξο λέγεται κεφαλή του κανόνα και δεξιά από το τόξο λέγεται σώμα του κανόνα. Αν το σώμα του κανόνα είναι αληθές τότε και η κεφαλή θα είναι αληθής και αντίστοιχα αν το σώμα είναι ψευδές τότε και η κεφαλή είναι ψευδής. Μπορούμε να έχουμε και κανόνες που να μην έχουν σώμα, να υπάρχει μόνο η κεφαλή του κανόνα. Αυτοί οι κανόνες ονομάζονται γεγονότα και είναι πάντα αληθείς.

Τα προγράμματα αυτά αποτελούνται από πεπερασμένα σύνολα από κανόνες. Σκοπός τους είναι να καθορίσουν αν είναι αληθείς ή ψευδείς τα atoms που χρειάζονται να καθοριστούν μέσα στο πρόγραμμα. Τα σύνολα απαντήσεων έρχονται για να δώσουν μία λύση η οποία είναι αποδεκτή.

Το παράδειγμα στο σχήμα 2.9 είναι ένα πρόγραμμα συνόλου απαντήσεων βάση του οποίου θα αναλύσουμε περισσότερο κάποια σημεία του συνόλου απαντήσεων.

high_salary ← employed, educated
educated ← high_salary
employed ← motivated
motivated.

Σχήμα 2.9 Παράδειγμα προγράμματος συνόλου απαντήσεων

Στο πρόγραμμα υπάρχουν 4 atoms που θα πρέπει στην λύση να καθοριστούν αν είναι αληθείς ή ψευδείς. Υπάρχουν επίσης 4 κανόνες, κάθε γραμμή είναι και ένας κανόνας στο παράδειγμα. Παρατηρούμε ότι το motivated είναι σε ένα κανόνα που πιο πριν εξηγήσαμε ότι είναι αυτός είναι ένας κανόνας που λέγεται γεγονός. Άρα το motivated είναι αληθείς. Βάση αυτού μπορούμε να καθορίσουμε ότι και το employed είναι αληθή βάση του 3^{ου} κανόνα. Για να καθοριστεί το high_salary χρειάζεται να καθοριστεί το educated και για να καθοριστεί το educated πρέπει να καθοριστεί το high_salary βάση των κανόνων 1 και 2. Αυτό το αδιέξοδο δεν μπορεί να σπάσει από την στιγμή που δεν υπάρχει άλλος κανόνας που να μας καθορίσει το high_salary ή το educated. Άρα με αυτούς τους κανόνες δεν γνωρίζουμε τι είναι το high_salary και το educated και το πρόγραμμα μπορεί να επιτρέψει σαν απάντηση μόνο το σύνολο {motivated , employed}.

open ← not closed
closed ← not open.

Σχήμα 2.10 Παράδειγμα 2 προγράμματος συνόλου απαντήσεων

Στο παράδειγμα του σχήματος 2.9 ήταν ξεκάθαρο το από πού θα ξεκινήσει και πως η επεξεργασία. Στο παράδειγμα του σχήματος 2.10 όμως δεν είναι ξεκάθαρο το από πού θα ξεκινήσει η επεξεργασία. Δεν μπορούμε να καθορίσουμε πιο από τα δύο atoms είναι αληθή για να εφαρμόσουμε τους κανόνες και να καταλήξουμε σε ένα αποτέλεσμα. Σε αυτές τις περιπτώσεις ξεκινούμε με το να υποθέσουμε πιο atom δεν είναι αληθή. Ας υποθέσουμε λοιπόν ότι το closed είναι ψευδή τότε μπορούμε να χρησιμοποιήσουμε τον $1^{ο}$ κανόνα και να καθορίσουμε το open ως αληθή. Το σύνολο {open} δικαιολογείται από το πρόγραμμα. Όσα atoms είναι μέσα στο αποδεκτό σύνολο σημαίνει ότι είναι αληθή και όσα δεν είναι μέσα σημαίνει ότι είναι ψευδή. Σε αυτό το παράδειγμα δεν είναι μόνο το σύνολο {open} αποδεκτό, μπορούμε να πούμε ότι το σύνολο {closed} είναι αποδεκτό αν ξεκινήσουμε στην αρχή με την υπόθεση ότι το open είναι ψευδή.

Με τα δύο παραδείγματα πιο πάνω καταλήγουμε στο συμπέρασμα ότι προγράμματα χωρίς not στο σώμα του κανόνα είναι πιο εύκολα στο να δώσουν ένα αποτέλεσμα που να είναι αποδεκτό. Δεν χρειάζεται να γίνει οποιαδήποτε υπόθεση για το τι θα είναι αληθή και τι θα είναι ψευδή για να καταλήξουμε σε κάποιο αποτέλεσμα. Σε κάθε βήμα χρησιμοποιούνται τα προηγούμενα atoms που έχουν καθοριστεί για να καθοριστούν καινούργια. Όταν πλέον δεν μπορούν να καθοριστούν νέα atoms τότε το πρόγραμμα τερματίζεται. Τα μοναδικά σύνολα από τα atoms που έχουν καθοριστεί λέγονται και ως το σύνολο απαντήσεων του προγράμματος.

Το θέμα στο παράδειγμα του σχήματος 2.10 είναι σημαντικό. Ξεκινούμε με ένα σύνολο M όπου μέσα είναι όλα τα atoms που μπορούν να καθοριστούν και κάνουμε κάποιες υποθέσεις για το ότι όσα atoms δεν βρίσκονται μέσα σε αυτό τότε δεν μπορούν να καθοριστούν. Στη συνέχεια κανόνες που περιέχουν στο σώμα τους το not a , όπου το a βρίσκεται μέσα στο M γίνονται άχρηστοι. Αυτοί οι κανόνες έχουν μπλοκαριστεί από το σύνολο M και μπορούν να αγνοηθούν. Όσοι κανόνες έχουν μέσα την άρνηση κάποιου ατομικού στοιχείου τότε θα πρέπει αυτό το στοιχείο να μην μπορεί να καθοριστεί (να είναι δηλαδή ψευδή) ή θα πρέπει ο κανόνας αυτός να διαγραφεί από το πρόγραμμα.

Κεφάλαιο 3

Τυχαία Γραφήματα

3.1 Βιβλιοθήκη python-igraph

Η python-igraph είναι μια βιβλιοθήκη της γλώσσας προγραμματισμού python που είναι υπεύθυνη για την δημιουργία όλων των γραφημάτων μέσα στο πρόγραμμα. Είναι μία βιβλιοθήκη ανοικτού πηγαίου κώδικα και μπορεί να την χρησιμοποιήσει όποιος θέλει. Μέσα σε αυτή την βιβλιοθήκη υπάρχουν υλοποιημένες μέθοδοι που δημιουργούν διάφορα είδη τυχαίων γραφημάτων. Στο πρόγραμμα που υλοποίησα χρησιμοποίησα 9 είδη από αυτά που υπάρχουν μέσα στη βιβλιοθήκη (Barabasi , Erdos-Renyi, K_Regular, Static Power Law, Forest Fire, Growing Random, Recent Degree, Static Fitness και Degree Sequence). Το κάθε γράφημα από αυτά χρειάζεται διαφορετικές παραμέτρους για να δημιουργηθεί, αυτό θα εξηγηθεί αργότερα. Μέσα από την βιβλιοθήκη αυτή μας δίνεται η δυνατότητα να προσθέσουμε ή να αφαιρέσουμε κορυφές σε ένα δημιουργημένο γράφο. Μπορούμε επίσης να προσθέσουμε ή να αφαιρέσουμε ακμές σε ένα δημιουργημένο γράφο.

Γενικά σε αυτή την βιβλιοθήκη υπάρχουν υλοποιημένες μέθοδοι με τις οποίες μπορεί κάποιος να επεξεργαστεί ένα γράφο.

3.2 Είδη τυχαίων γραφημάτων

Barabasi Graph

Η βιβλιοθήκη python-graph[10] για να δημιουργήσει ένα γράφο Barabasi χρησιμοποιεί την μέθοδο

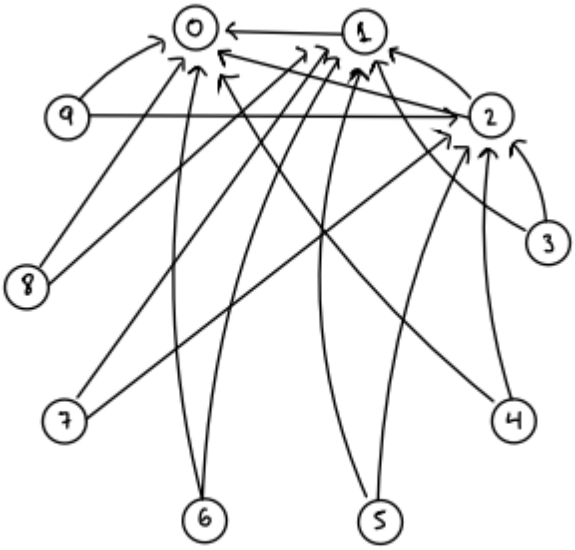
def Barabasi (n, m, directed=True)

όπου n είναι οι κορυφές που θα έχει ο γράφος, m θα είναι οι ακμές που θα έχει η κάθε κορυφή και directed = True αν είναι κατευθυνόμενος ο γράφος ή directed = False αν είναι μη κατευθυνόμενος ο γράφος που θέλουμε να δημιουργήσουμε.

Η μέθοδος αυτή δημιουργεί ένα γράφο ο οποίος είναι βασισμένος στο Barabasi-Albert μοντέλο. [5] Το Barabasi-Albert μοντέλο είναι ένας αλγόριθμος για δημιουργία τυχαίων μη κλιμακωμένων δικτύων (scale-free networks) χρησιμοποιώντας ένα προνομιακό επισυναπτόμενο μηχανισμό (preferential attachment mechanism).

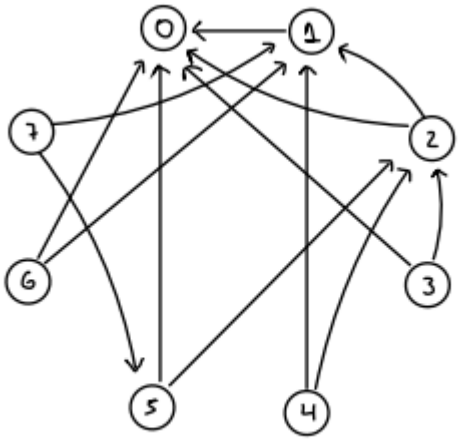
Αυτό το μοντέλο προσπαθεί να εξηγήσει την ύπαρξη πολλών κόμβων σε πραγματικά δίκτυα. Ο αλγόριθμος πήρε το όνομα του από τους δημιουργούς του Albert-Laszlo Barabasi και Reka Albert και είναι μία ειδική περίπτωση ενός πιο γενικού μοντέλου το Price’s model. Πολλά δίκτυα που έχουν παρατηρηθεί ανήκουν στην κλάση των μη κλιμακούμενων δικτύων, αυτό σημαίνει ότι έχουν power-law κατανομή βαθμού. Ο Barabasi είναι ένα από πολλά μοντέλα που προτάθηκαν τα οποία δημιουργούν μη κλιμακούμενα δίκτυα.

Η μέθοδος αυτή δημιουργεί ένα γράφο με τον εξής τρόπο. Ξεκινά με την δημιουργία της πρώτης κορυφής (κορυφή 0), στη συνέχεια ενώνει την κορυφή αυτή με m τυχαίες κορυφές που έχουν αριθμό ταυτοποίησης μικρότερο από τον αριθμό της κορυφής αυτής. Όπου m είναι ο αριθμός που δίνεται στην μέθοδο και σημαίνει πόσες ακμές θα έχει η κάθε κορυφή. Στη συνέχεια δημιουργείται η επόμενη κορυφή και ενώνεται με m κορυφές με μικρότερο αριθμό. Αυτό συνεχίζεται μέχρι να δημιουργήσει όλες τις n κορυφές που δίνονται σαν είσοδο στην μέθοδο.



Σχήμα 3.1 Barabasi Graph με 10 κορυφές και 2 ακμές

Στο σχήμα 3.1 είναι ένα παράδειγμα Barabasi graph με 10 κορυφές και 2 ακμές. Παρατηρούμε ότι από την κάθε κορυφή φεύγουν 2 ακμές εκτός από τις κορυφές 0 και 1. Από την κορυφή 0 δεν φεύγει κάποια ακμή επειδή δεν υπάρχει κορυφή με μικρότερο αριθμό από το 0. Από την κορυφή 1 φεύγει μόνο μία ακμή προς την κορυφή 0 επειδή μόνο μία κορυφή έχει μικρότερο αριθμό από την κορυφή 1.



Σχήμα 3.2 Barabasi Graph με 8 κορυφές και 2 ακμές

Όπως και στο σχήμα 3.1 έτσι και στο σχήμα 3.2 παρατηρούμε τα ίδια πράγματα. Δεν φεύγουν από την κορυφή 0 ακμές προς άλλες κορυφές και από την κορυφή 1 φεύγει μόνο μία ακμή προς την κορυφή 0.

Erdos-Renyi Graph

Το μοντέλο Erdos-Renyi[6] είναι ουσιαστικά δύο πολύ σχετικά μοντέλα για δημιουργία τυχαίων γράφων ή η εξέλιξη ενός τυχαίου δικτύου. Και τα δύο αυτά μοντέλα πήραν το όνομα τους από τους Paul Erdos και Alfred Renyi, οι οποίοι πρώτοι δημιούργησαν ένα από αυτά τα δύο μοντέλα το 1959, όπου ο Edgar Gilbert δημιούργησε το άλλο μοντέλο ανεξάρτητα από τον Erdos και τον Renyi.

Στο μοντέλο αυτών των δύο, όλοι οι γράφοι έχουν συγκεκριμένο αριθμό από κόμβους με συγκεκριμένο αριθμό από ακμές. Όλες οι κορυφές έχουν ακριβώς τον ίδιο αριθμό από ακμές. Στο μοντέλο του Gilbert, η κάθε ακμή έχει ακριβώς την ίδια πιθανότητα για να παρουσιαστεί μέσα στο γράφο.

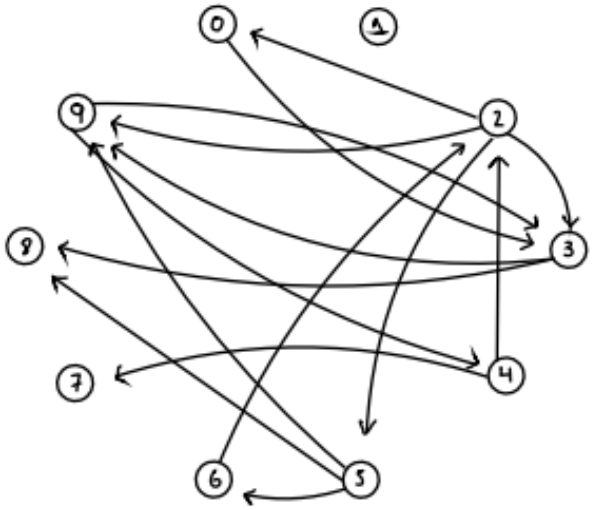
Στο πρώτο μοντέλο G(n,M), στο οποίο ο αριθμός των ακμών , M , είναι συγκεκριμένος και δίνεται σαν είσοδος στον γράφο, ο γράφος που τελικά θα δημιουργηθεί διαλέγεται μέσα από ένα σύνολο από γράφους που έχουν n κορυφές και M ακμές, Για παράδειγμα στο μοντέλο G(3,2) , υπάρχουν τρεις γράφοι που έχουν 2 ακμές με 3 κορυφές και ο κάθε ένας από αυτούς τους τρεις γράφους έχει πιθανότητα για να δημιουργηθεί 1/3.Όλοι οι γράφοι έχουν την ίδια πιθανότητα.

Στο δεύτερο μοντέλο G(n,p), ο γράφος κατασκευάζεται με το να ενώνονται οι κορυφές τυχαία. Κάθε ακμή περιλαμβάνεται στο γράφο με πιθανότητα p, ανεξάρτητα από τις υπόλοιπες ακμές. Η πιθανότητα για δημιουργία κάθε γράφου που έχει n κόμβους και M ακμές δίνεται από την σχέση στο σχήμα 3.3.

$$p^M (1-p)^{\binom{n}{2}-M}.$$

Σχήμα 3.3 Πιθανότητα γράφου στο δεύτερο μοντέλο

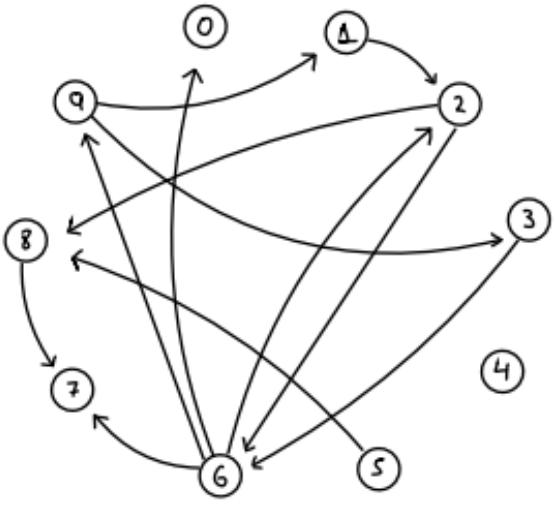
Η πιθανότητα p στο μοντέλο μπορεί να γίνει κατανοητή ως μέθοδος βάρους. Οι πιθανές τιμές που μπορεί να πάρει το p είναι από 0 μέχρι 1. Όσο πιο μεγάλη είναι η πιθανότητα αυτή τόσο πιο πολλές ακμές θα έχει ο γράφος που θα δημιουργηθεί.



Σχήμα 3.4 Erdos-Renyi Graph G(10,16)

Στο σχήμα 3.4 είναι ένας Erdos-Renyi γράφος με 10 κορυφές και 16 ακμές. Ο γράφος αυτός δημιουργήθηκε με την μέθοδο της βιβλιοθήκης python-igraph

def Erdos_Renyi (10, m=16, directed=False)



Σχήμα 3.5 Erdos-Renyi Graph G(10, 0.15)

Στο σχήμα 3.5 είναι ένας άλλος γράφος που βασίζεται στο μοντέλο του Erdos-Renyi αλλά είναι βάση του δεύτερου μοντέλου που ανέφερα πιο πάνω. Αυτός ο γράφος δημιουργήθηκε με την μέθοδο

def Erdos_Renyi (10, 0,15, directed=False) [10]

K_Regular Graph

Στη βιβλιοθήκη python-igraph[10] την οποία χρησιμοποιώ στο πρόγραμμα που υλοποίησα έχει μέθοδο και για την δημιουργία k_regular γράφου. Η μέθοδος αυτή είναι

def K_Regular (n, k, directed=False)

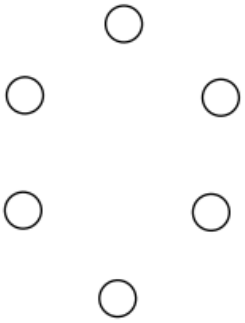
όπου n είναι οι κορυφές του γράφου και k είναι ο βαθμός της κάθε κορυφής.

Στη θεωρία των γράφων, ένας κανονικός(regular)[7] γράφος είναι ο γράφος που η κάθε κορυφή του έχει ακριβώς τους ίδιους γείτονες με τις υπόλοιπες κορυφές του γράφου. Ένας κατευθυνόμενος κανονικός γράφος πρέπει να ικανοποιεί και τον περιορισμό ότι οι ακμές που έρχονται σε μια κορυφή πρέπει να είναι ο ίδιος με τον αριθμό τον ακμών που κατευθύνονται σε όλες τις άλλες κορυφές. Επίσης και ο αριθμός των ακμών που φεύγουν από μια κορυφή σε αυτό τον γράφο πρέπει να είναι ο ίδιος με τον αριθμό των ακμών που φεύγουν από τις υπόλοιπες κορυφές του γράφου.

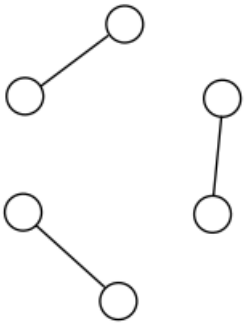
Για να μπορεί να δημιουργηθεί ένας k_regular γράφος με n κορυφές θα πρέπει να ισχύουν κάποιοι περιορισμοί

1. Ο αριθμός των κορυφών του γράφου θα πρέπει να είναι μεγαλύτερος ή ίσος με το k+1, όπου k είναι ο βαθμός του γράφου.
2. Ο αριθμός των κορυφών επί τον βαθμό του γράφου θα πρέπει να είναι ζυγός αριθμός.

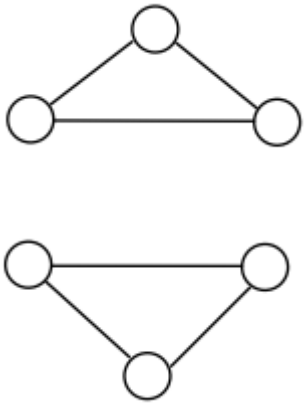
Ένας κανονικός γράφος με βαθμό το πολύ 2 είναι εύκολο να ταξινομηθεί . Ένας 0-regular γράφος αποτελείτε από μη ενωμένες κορυφές μόνο, ένας 1-regular γράφος αποτελείται από ακμές που δεν είναι ενωμένες και ένας 2-regular γράφος αποτελείται από την ένωση των κύκλων που υπάρχουν στον γράφο αλλά δεν είναι ενωμένοι. Ένας 3-regular γράφος είναι γνωστός και ως κυβικός γράφος.



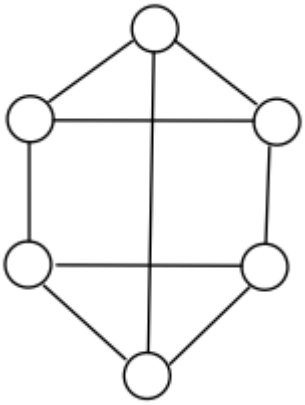
Σχήμα 3.6 0-regular graph



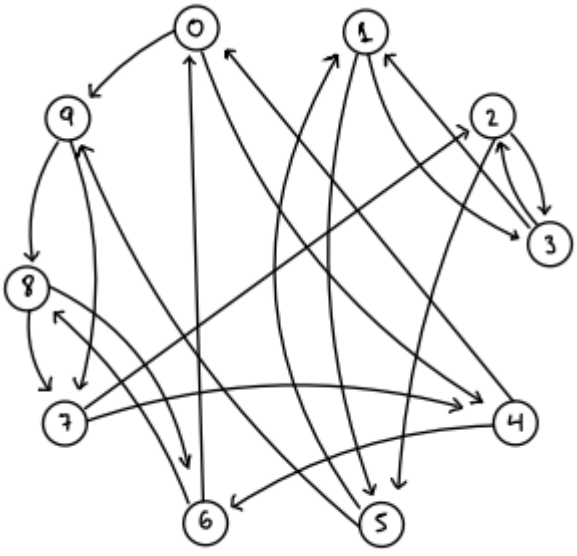
Σχήμα 3.7 1-regular graph



Σχήμα 3.8 2-regular graph

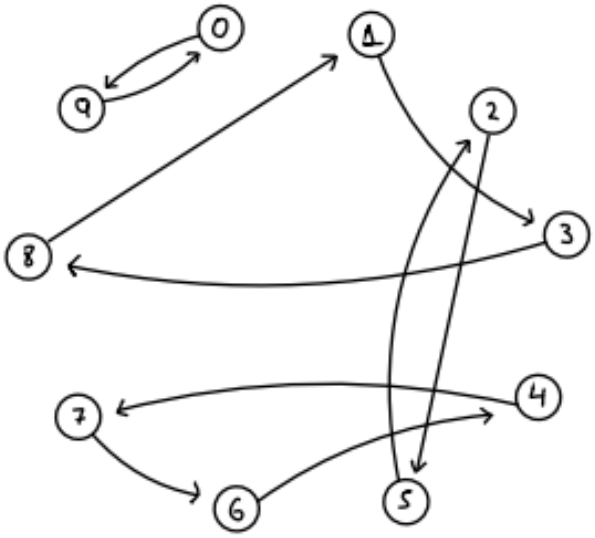


Σχήμα 3.9 3-regular graph



Σχήμα 3.10 K_Regular γράφος με 10 κορυφές και βαθμό 2

Στο σχήμα 3.10 παρατηρούμε ότι κάθε κορυφή έχει τον ίδιο αριθμό ακμών που φεύγουν από αυτή και επίσης έχουν τον ίδιο αριθμό ακμών που έρχονται σε κάθε κορυφή. Ο αριθμός των ακμών αυτών καθορίζεται από το βαθμό του γράφου. Στο παράδειγμα του σχήματος 3.10 ο βαθμός του γράφου είναι 2 για αυτό τον λόγο ο αριθμός των ακμών που φεύγουν από μία κορυφή και ο αριθμός των ακμών που έρχονται σε μία κορυφή είναι 2.



Σχήμα 3.11 K_Regular γράφος με 10 κορυφές και βαθμό 1

Στο σχήμα 3.11 παρατηρούμε όπως και πριν ότι κάθε από κάθε κορυφή φεύγει ο ίδιος αριθμός ακμών και έρχεται στην κάθε κορυφή ο ίδιος αριθμός ακμών. Στο παράδειγμα αυτό ο βαθμός του γράφου είναι 1, άρα ο αριθμός των ακμών που θα φεύγουν και θα έρχονται σε κάθε κορυφή είναι 1.

Static Power Law Graph

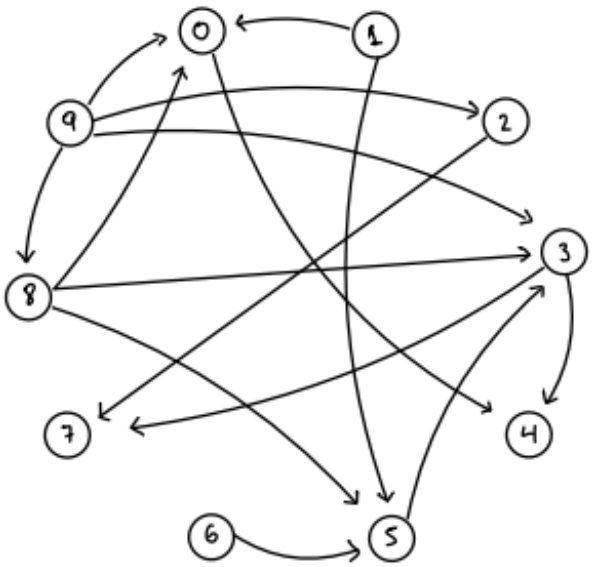
Υπάρχουν δύο σημαντικά συστατικά τα οποία εξηγούν την εμφάνιση της μη-κλιμακούμενης ιδιότητας(scale-free property) σε ένα περίπλοκο δίκτυο, η ανάπτυξη και οι προτιμώμενες επισυνάψεις(preferential attachment). Με την ανάπτυξη εννοούμε ότι κάθε ένα διάστημα χρόνου καινούργιοι κόμβοι δημιουργούνται και θέλουν να είναι μέρος του δικτύου. Με τις προτιμώμενες επισυνάψεις εννοούμε ότι ένας νέος κόμβος θέλει να ενωθεί με άλλους υφιστάμενους κόμβους που ήδη έχουν ένα αριθμό από γείτονες.

Υπάρχει μεγαλύτερη πιθανότητα ότι περισσότεροι κόμβοι θα ενώνονται με εκείνους που έχουν περισσότερες ενώσεις με άλλους κόμβους, αυτό ονομάζεται hub. Εξαρτάτε από το δίκτυο, τα hubs μπορεί να είναι ανάλογα και δυσανάλογα. Ανάλογα μπορεί να βρεθούν σε κοινωνικά δίκτυα όπου διάσημοι μπορεί να γνωρίζουν καλύτερα ο ένας τον άλλον και δυσανάλογα μπορεί να βρεθούν σε τεχνολογικά και βιολογικά δίκτυα.

Στη βιβλιοθήκη python-igraph[10] υπάρχει μέθοδος που δημιουργεί static power law γράφο

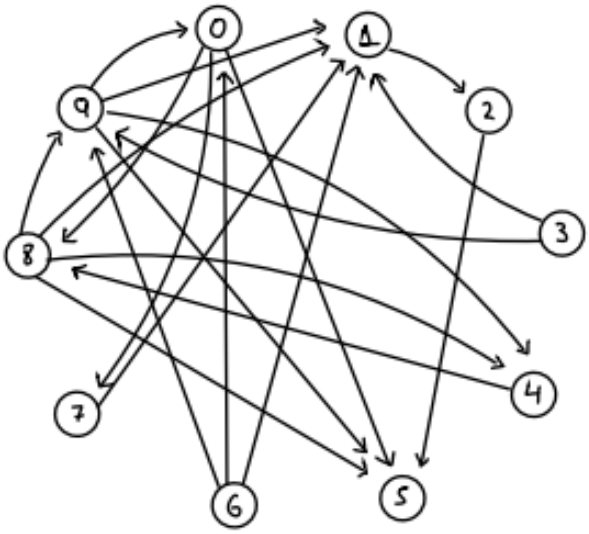
def Static_Power_Law(n, m, exponent_out, exponent_in=-1, loops=False, multiple=False)

- n είναι ο αριθμός των κορυφών
- m είναι οι ακμές που θα έχει ο γράφος
- exponent_out είναι η κατανομή βαθμού που έχουν οι εξερχόμενες ακμές
- exponent_in είναι η κατανομή βαθμού που έχουν οι εισερχόμενες ακμές
- loops είναι True αν επιτρέπεται να υπάρχουν ακμές που κατευθύνονται στην ίδια κορυφή
- multiple είναι True αν γίνεται να υπάρχουν πολλαπλές ακμές



Σχήμα 3.12 Static Power Law Γράφος

Στο σχήμα 3.12 ο γράφος δημιουργήθηκε με τις εξής παραμέτρους, n=10, m=15, exponent_out=2, exponent_in=2.



Σχήμα 3.13 Static Power Law γράφος

Ο γράφος στο σχήμα 3.13 δημιουργήθηκε με n=10, m=20,exponent_out=4, exponent_in=3.

Forest Fire Graph

Το forest fire[8] μοντέλο είναι ορισμένο ως το κυτταρικό αυτόματο του πλέγματος με L_d κελιά. L είναι το πλευρικό μήκος του πλέγματος και d είναι η διάσταση του. Ένα κελί μπορεί να είναι κενό, μπορεί να έχει ένα δέντρο ή μπορεί να καίγεται. Οι ελεγχόμενες παράμετροι του μοντέλου είναι το p και το f, τα οποία μας δίνουν τον μέσο όρο αριθμών δέντρων που φυτεύονται μεταξύ δύο αστραπών.

Το μοντέλο των Drossel και Schwabl είναι ορισμένο από τέσσερις κανόνες οι οποίοι εκτελούνται ταυτόχρονα:

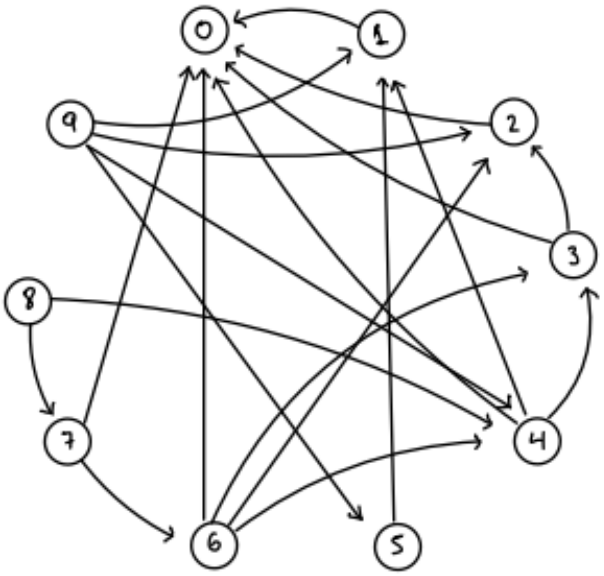
1. Ένα φλεγόμενο κελί μετατρέπεται σε ένα κενό κελί.
2. Ένα δέντρο θα καίγεται αν τουλάχιστον ένας γείτονας του καίγεται.
3. Ένα δέντρο αναφλέγεται με πιθανότητα f ακόμη και αν κανένας γείτονας του δεν καίγεται.
4. Ένα κενό κελί γεμίζεται με δέντρο με πιθανότητα p

Αυτό το μοντέλο ξεκινά με μια μη ελεγχόμενη ανάπτυξη δέντρων. Μετά από λίγη ώρα, μια αστραπή θα ξεκινήσει την φωτιά. Η φωτιά θα εξαπλωθεί καταστρέφοντας δέντρα σε μεγάλες εκτάσεις. Ταυτόχρονα, στο σημείο της φωτιάς, θα μεγαλώσουν καινούργια δέντρα. Αν έχουμε μία πιθανότητα δημιουργίας δέντρων p και μία πιθανότητα φωτιάς q ορισμένη σε λογικά επίπεδα, μπορούμε να δούμε μία ομάδα από δέντρα που μεγάλωσαν και μία ομάδα από φλεγόμενα δέντρα. Διαφορετικά θα έχουμε μία τυχαία κατανομή από κενά κελιά, κελιά που θα έχουν δέντρα και κελιά που θα φλέγονται.

Στο πρόγραμμα που υλοποίησα για να δημιουργήσω ένα τέτοιο γράφημα χρησιμοποίησα την μέθοδο της βιβλιοθήκη python-igraph[10]

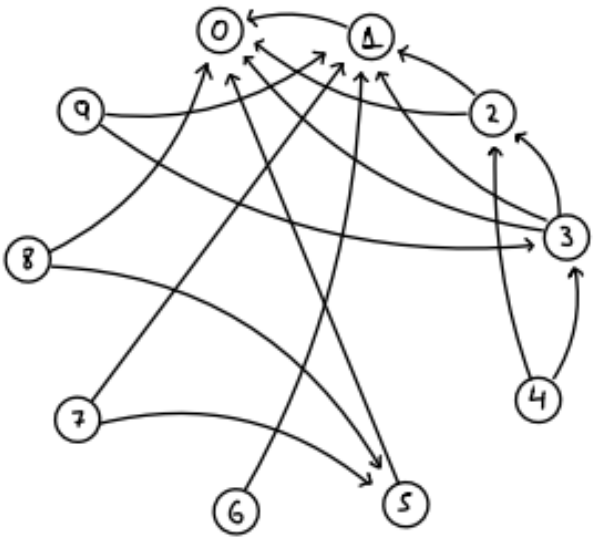
def Forest_Fire(n, fw_prob, bw_factor=0.0, ambs=1, directed=False)

όπου, n είναι οι κορυφές του γράφου, fw_prob είναι η πιθανότητα καύσης προς τα εμπρός, bw_factor είναι η αναλογία καύσης προς τα εμπρός και προς τα πίσω και ambs(ambassadors) είναι ο αριθμός των πρεσβευτών σε κάθε βήμα.



Σχήμα 3.14 Forest Fire Graph με 10 κορυφές

Στο σχήμα 3.14 ο γράφος αυτός δημιουργήθηκε με την μέθοδο που ανέφερα πιο πάνω. Στις παραμέτρους έδωσα τις τιμές, n=10, fw_prob=0.5, bw_factor=0.5, ambs=2.



Σχήμα 3.15 Forest Fire Graph με 10 κορυφές

Στο σχήμα 3.15 έδωσα τις τιμές n=10, fw_prob=0.3, bw_factor=0.3, ambs=2.

Growing Random Graph

Σε αυτό το είδος του γράφου προστίθεται μία κορυφή[9] κάθε βήμα και αυτή η κορυφή ενώνεται με διάφορες άλλες κορυφές που είναι ήδη δημιουργημένες. Αν υποθέσουμε ένα γράφο του διαδικτύου, οι κορυφές είναι οι ιστοσελίδες και οι κατευθυνόμενες ακμές είναι σύνδεσμοι που ενώνουν την μία ιστοσελίδα με την άλλη. Αναφερόμαστε στον κόμβο που κατευθύνετε η ακμή ως τον πρόγονο του κόμβου που ξεκινά η ακμή.

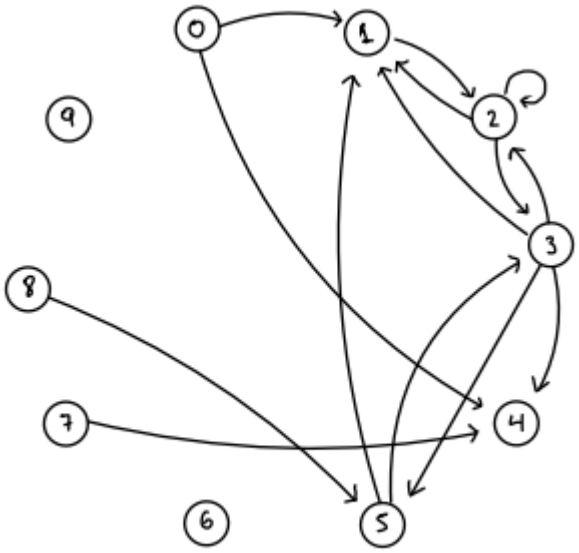
Όπως μεγαλώνει το δίκτυο, η κατανομή βαθμού ορίζεται ως τον μέσο αριθμό κόμβων με κ συνδέσμους. Η πρώτη κορυφή είναι μοναδική επειδή δεν έχει ακμές που φεύγουν από αυτή. Το βασικό συστατικό το οποίο καθορίζει την δομή του δικτύου είναι ο επισυναπτόμενος πυρήνας, ο οποίος ορίζεται ως την πιθανότητα με την οποία οι καινούργιοι κόμβοι ενώνονται με κόμβους που υπάρχουν και ήδη έχουν κ συνδέσμους. Γενικά, αυτός ο επισυναπτόμενος πυρήνας πρέπει να είναι μία μη-αυξανόμενη μέθοδος του κ.

[10] Στο πρόγραμμα που υλοποίησα για να δημιουργήσω τέτοιου είδους γράφους χρησιμοποιώ μία μέθοδο της βιβλιοθήκης python-igraph

def Growing_Random(n, m, directed=False)

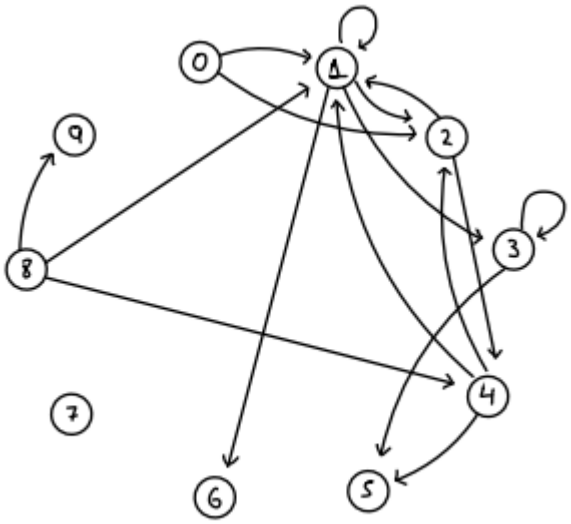
όπου,

- n είναι οι κορυφές του γράφου που θα δημιουργηθεί
- m είναι ο αριθμός των ακμών που θα προστεθούν σε κάθε βήμα
- directed είναι αν ο γράφος θα είναι κατευθυνόμενος ή όχι



Σχήμα 3.16 Growing Random γράφος με 10 κορυφές

Στο παράδειγμα του σχήματος 3.16 οι παράμετροι που δόθηκαν για την δημιουργία αυτού του γράφου είναι n=10 και m=2



Σχήμα 3.17 Growing Random γράφος με 10 κορυφές

Στο παράδειγμα του σχήματος 3.17 δημιουργήθηκε ένας γράφος growing random με 10 κορυφές και m=3.

Recent Degree Graph

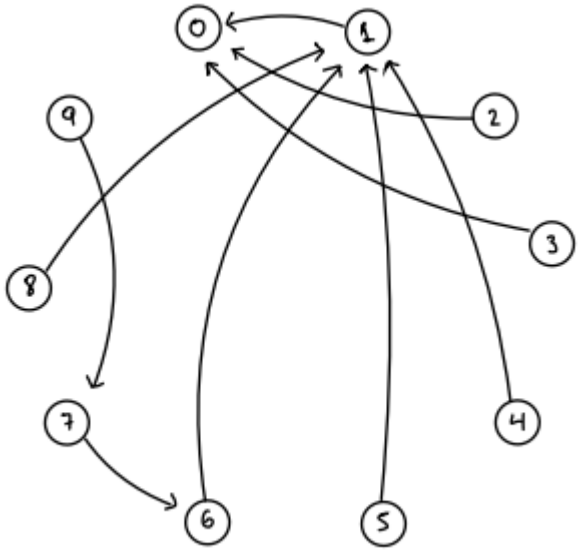
Αυτός ο γράφος είναι βασισμένος σε ένα στοχαστικό μοντέλο[10] στο οποίο η πιθανότητα μιας ακμής να ενωθεί με κάποιο νέο κόμβο είναι ανάλογη των ακμών που ενώνονται σε ένα συγκεκριμένο περιθώριο χρόνου.

Η μέθοδος που χρησιμοποιώ για την παραγωγή αυτού του γραφήματος είναι

def Recent_Degree(n, m, window, outpref=False, directed=False, power=1):

όπου

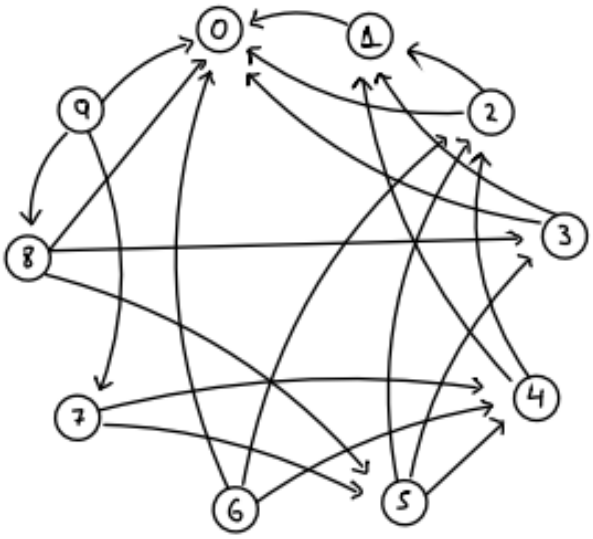
- n είναι ο αριθμός των κορυφών που θα έχει ο γράφος
- m είναι ο αριθμός τω εξερχόμενων ακμών που δημιουργούνται για κάθε κορυφή
- window είναι το μέγεθος του παραθύρου σε βήματα χρόνου
- outpref είναι True αν το index τον κορυφών που ενώνονται με κάθε κορυφή που δημιουργείτε είναι αυξανόμενο, διαφορετικά ενώνονται πάντα με το 0
- directed είναι True αν ο γράφος είναι κατευθυνόμενος
- Power είναι η σταθερά ισχύος ενός μη γραμμικού μοντέλου.



Σχήμα 3.18 Recent Degree γράφος με 10 κορυφές

Στο παράδειγμα του σχήματος 3.18 ο γράφος δημιουργήθηκε με τις εξής παραμέτρους.

- n=10
- m=1
- window=2
- outpref=True
- directed=True



Σχήμα 3.19 Recent Degree γράφος με 10 κορυφές

Στο παράδειγμα του σχήματος 3.19 ο γράφος δημιουργήθηκε με τις εξής παραμέτρους.

- n=10
- m=3
- window=2
- outpref=True
- directed=True

Static Fitness Graph

Το μοντέλο αυτό είναι βασισμένο στην ιδέα της καταλληλότητας, ενός έμφυτου ανταγωνιστικού παράγοντα τον οποίο οι κόμβοι μπορεί να έχουν, ικανός να επιδράσει την εξέλιξη του δικτύου. Βάση αυτής της ιδέας, η ικανότητα των κόμβων να προσελκύουν συνδέσμους στο δίκτυο μέσα από διάφορους κόμβους. Ο πιο αποδοτικός θα μπορεί να κερδίσει περισσότερες ακμές από τους υπόλοιπους. Με αυτή την έννοια δεν είναι όλοι οι κόμβοι πανομοιότυποι με τους υπόλοιπους, και παίρνουν την αύξηση του βαθμού τους σύμφωνα με την καταλληλότητα που επεξεργάζονται κάθε στιγμή.

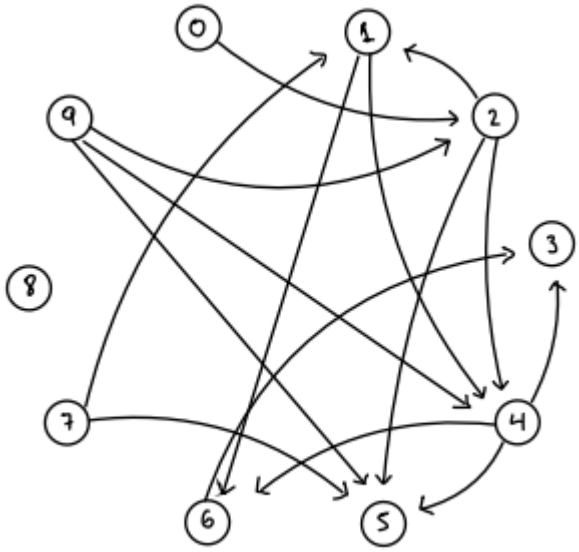
Ο παράγοντας καταλληλότητας για όλους τους κόμβους που συνθέτει το δίκτυο μπορεί να ακολουθεί μία κατανομή χαρακτηριστική από το σύστημα που μελετήθηκε.

Στο πρόγραμμα που υλοποίησα χρησιμοποιώ την μέθοδο της python-igraph

def Static_Fitness(m, fitness_out, fitness_in=None, loops=False, multiple=False):

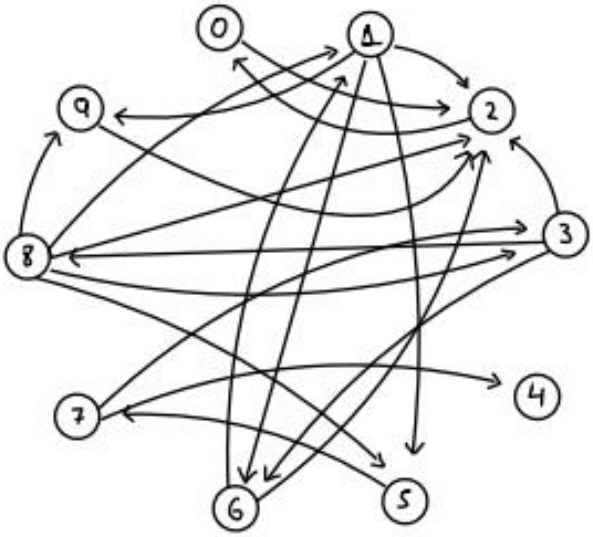
- m είναι ο αριθμός των ακμών που θα υπάρχουν στον γράφο
- fitness_out είναι μία λίστα από μη-αρνητικές τιμές που αντιπροσωπεύουν την καταλληλότητα για τις εξερχόμενες ακμές της κάθε κορυφής αντίστοιχα
- fitness_in είναι μία λίστα από μη αρνητικές τιμές που αντιπροσωπεύουν την καταλληλότητα για τις εισερχόμενες ακμές τις κάθε κορυφής αντίστοιχα
- loops είναι True όταν επιτρέπεται οι ακμές να κατευθύνονται στην ίδια κορυφή
- multiple είναι True αν επιτρέπονται η πολλαπλές ακμές.

Αυτή η μέθοδος δημιουργεί ένα μη-αναπτυσσόμενο γράφο με πιθανότητα ακμής ανάλογη της καταλληλότητας του κόμβου. Ο αλγόριθμος τυχαία επιλέγει ζεύγη κορυφών και τα ενώνει να δημιουργηθεί ο αριθμός των ακμών που δόθηκαν. Κάθε κορυφή διαλέγεται με πιθανότητα ανάλογη στην καταλληλότητα της, στους κατευθυνόμενους γράφους, μία κορυφή επιλέγεται σαν η πηγή της ακμής ανάλογα με την καταλληλότητα για τις εξερχόμενες ακμές που έχει και μία άλλη ακμή επιλέγεται σαν ο στόχος την ακμής ανάλογα με την καταλληλότητα για τις εισερχόμενες ακμές που έχει.



Σχήμα 3.20 Static Fitness Γράφος

Στο παράδειγμα του σχήματος 3.20 ο γράφος αυτός δημιουργήθηκε με παραμέτρους m=15, fitness_out=[3,4,6,2,7,2,5,8,1,7] και fitness_in=[3,7,3,8,4,9,6,3,1,6].



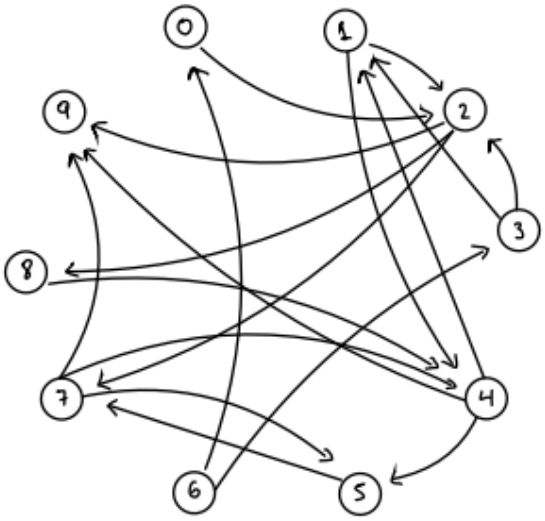
Σχήμα 3.21 Static Fitness Γράφος

Στο παράδειγμα του σχήματος 3.21 ο γράφος αυτός δημιουργήθηκε με παραμέτρους m=20, fitness_out=[5,4,7,3,1,3,5,8,9,2] και fitness_in=[2,4,9,3,6,7,4,1,3,5]

Degree Sequence Graph

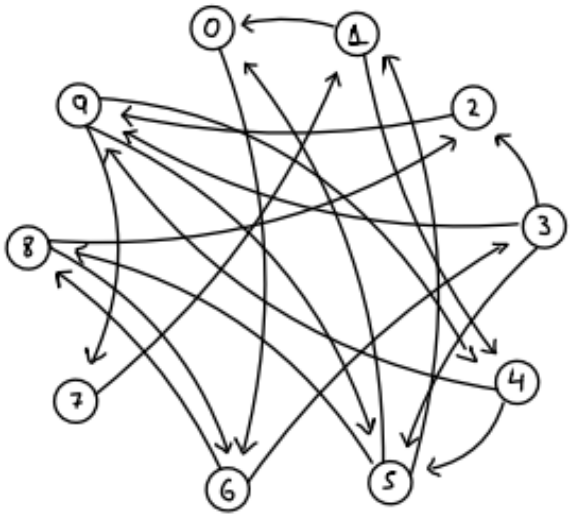
[10] Το είδος του γράφου αυτού είναι σχετικά απλό. Για να δημιουργηθεί αυτός ο γράφος χρειάζεται μόνο η ακολουθία βαθμών των εισερχόμενων ακμών τις κάθε κορυφής και η ακολουθία των βαθμών των εξερχόμενων ακμών. Δηλαδή πρέπει να δοθεί για κάθε κορυφή πόσες ακμές θα εισέρχονται και θα εξέρχονται σε αυτή. Επιπλέον δίνεται μία παράμετρος που ονομάζεται μέθοδος. Υπάρχουν τριών ειδών μέθοδοι για την δημιουργία τέτοιου είδους γράφου στην βιβλιοθήκη python-igraph.

- Simple – δημιουργεί με απλό τρόπο τον γράφο. Μπορεί να υπάρχουν ακμές που φεύγουν και έρχονται στην ίδια ακμή και μπορεί να υπάρχουν και πολλαπλές ακμές.
- No_multiple – είναι παρόμοιο με το simple αλλά δημιουργεί γράφους που δεν έχουν πολλαπλές ακμές και δεν υπάρχουν ακμές που να ξεκινούν και να κατευθύνονται στην ίδια ακμή. Η μέθοδος αυτή ξεκινά ξανά από την αρχή την δημιουργία του γράφου κάθε φορά που φτάνει σε αδιέξοδο στο οποίο δεν μπορεί να προσθέσει άλλες ακμές και πρέπει να δημιουργήσει πολλαπλές ακμές ή ακμές που θα ξεκινούν και να κατευθύνονται στην ίδια κορυφή. Δεν υπάρχει όριο στις προσπάθειες που θα κάνει
- vl – μια πιο σοφιστική μέθοδος δημιουργίας γράφων η οποία χρησιμοποιείται μόνο για την δημιουργία μη κατευθυνόμενων γράφων. Χρησιμοποιεί Monte-Carlo μεθόδους για να τυχαιοποιήσει τον γράφο.



Σχήμα 3.22 Degree Sequence Γράφος

Στο παράδειγμα του σχήματος 3.22 οι παράμετροι που δόθηκαν για την δημιουργία του γράφου είναι out_degree = [1,2,3,2,3,1,2,3,1,0], in_degree=[1,2,3,1,3,2,0,2,1,3] και method=”no_multiple”.



Σχήμα 3.23 Degree Sequence Γράφος

Στο παράδειγμα του σχήματος 3.22 οι παράμετροι που δόθηκαν για την δημιουργία του γράφου είναι out_degree = [1,2,1,3,2,3,2,1,2,3], in_degree=[2,2,2,1,2,3,2,1,2,3] και method=”no_multiple”.

Κεφάλαιο 4

Παραγωγή Θεωριών Επιχειρηματολογίας

4.1 Δημιουργία γραφημάτων

Το πρόγραμμα ξεκινά με την δημιουργία του υπεργράφου ανάλογα με το είδος που θα επιλέξει ο χρήστης. Ο χρήστης θα δώσει τον αριθμό των κορυφών και τα υπόλοιπα χαρακτηριστικά που χρειάζεται το κάθε είδος γράφου που επέλεξε ο χρήστης να δημιουργήσει. Στη συνέχεια το πρόγραμμα δημιουργεί ένα υπογράφο για κάθε μία από τις κορυφές του υπεργράφου. Στη συνέχεια το πρόγραμμα ενώνει του υπογράφους και για κάθε ακμή που υπάρχει στον υπεργράφο ενώνει τα αντίστοιχα υπογραφήματα.

Για να λειτουργήσει σωστά το πρόγραμμα που υλοποίησα θα πρέπει να υπάρχουν στον ίδιο φάκελο με το πρόγραμμα, ένα αρχείο input.txt το οποίο μέσα θα υπάρχουν οι πληροφορίες γενικά για το τι είδος θα είναι ο υπογράφος, τι είδος θα είναι ο υπογράφος, με πόσες ακμές θα ενώνονται οι υπογράφοι και το ποσοστό των μη-κατευθυνόμενων ακμών.

```
hypergraph:2
hypergraphfile:erdos.txt
edgesbetweensub:2
subgraph:2
subgraphfile:erdossub.txt
percentage:10
```

Σχήμα 4.1 Περιεχόμενα αρχείου input.txt

Στην εικόνα του σχήματος 4.1 παρατηρούμε κάποιες πληροφορίες που πρέπει να δοθούν στο πρόγραμμα αρχικά για να δουλέψει σωστά.

Στο hypergraph και στο subgraph θα πρέπει να δοθεί ένας αριθμός από το 1 μέχρι και το 9, ο κάθε αριθμός αντιστοιχεί σε ένα είδος τυχαίου γραφήματος από την λίστα:

1. Barabasi Graph
2. Erdos-Renyi Graph
3. K_Regular Graph
4. Static Power Law Graph
5. Forest Fire Graph
6. Growing Random Graph
7. Recent Degree Graph
8. Static Fitness Graph
9. Degree Sequence Graph

Το hypergraphfile και το subgraphfile είναι τα αρχεία εισόδου για το κάθε είδος γράφο. Το κάθε είδος πρέπει να έχει το δικό του αρχείο εισόδου επειδή οι αντίστοιχες μέθοδοι που τους δημιουργούν χρειάζονται διαφορετικές παραμέτρους για να τους δημιουργήσουν. Το edgesbetweensub είναι ο μέγιστος αριθμός ακμών που θα είναι ενωμένοι οι υπογράφοι μεταξύ τους και τέλος το percentage είναι το ποσοστό των μη-κατευθυνόμενων ακμών που θα υπάρχουν στον τελικό γράφο.

```
vertices:5
edges:3-5
```

Σχήμα 4.2 Barabasi input file

Στο σχήμα 4.2 είναι οι πληροφορίες που χρειάζεται να έχει το αρχείο εισόδου του Barabasi γράφου. Το vertices είναι ο αριθμός των κορυφών που θα έχει ο γράφος και το edges είναι ο αριθμός των ακμών που θα έχει ο γράφος. Στο πρόγραμμα θα πρέπει να δίνει ο χρήστης ένα διάστημα από αριθμούς για να είναι πιο τυχαία η δημιουργία των διαφορετικών γράφων κάθε φορά.

```
vertices:5
probability:0.48
```

Σχήμα 4.3 Erdos-Renyi input file

Στο σχήμα 4.3 βλέπουμε τις πληροφορίες που χρειάζεται το πρόγραμμα για να δημιουργήσει ένα Erdos-Renyi γράφημα. Το vertices είναι ο αριθμός των κορυφών που θα έχει ο γράφος και το probability είναι η πιθανότητα των ακμών του γράφου.

```
vertices: 5
degree: 3
```

Σχήμα 4.4 K_Regular input file

Στο σχήμα 4.4 είναι οι πληροφορίες που χρειάζεται το πρόγραμμα για να δημιουργήσει ένα γράφημα K_Regular. Το vertices είναι ο αριθμός των κορυφών του γράφου και το degree είναι ο βαθμός του γράφου.

```
vertices:200
edges:10000
ex_out:200
ex_in:200
```

Σχήμα 4.5 Static Power Law input file

Στο σχήμα 4.5 είναι οι πληροφορίες που χρειάζεται το πρόγραμμα για να δημιουργήσει ένα Static Power Law γράφημα. Το vertices είναι ο αριθμός των κορυφών που θα έχει ο γράφος, το edges είναι ο αριθμός των ακμών του γράφου, το ex_out είναι ο αριθμός του exponent_out και το ex_in είναι ο αριθμός του exponent_in.

```
vertices:4
fw_prob:0.5
bw_factor:1
ambs:2
```

Σχήμα 4.6 Forest Fire input file

Στην περίπτωση του σχήματος 4.6 είναι οι πληροφορίες που χρειάζονται για την δημιουργία ενός forest fire γράφου. Το vertices είναι ο αριθμός των κορυφών που θα έχει ο γράφος, το fw_prob είναι το forward probability, το bw_factor είναι το backward factors και το ambs είναι ο αριθμός των ambassadors.

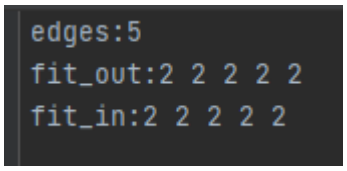
```
vertices:4
edges:2
```

Σχήμα 4.7 Growing Random input file

Στο σχήμα 4.7 είναι οι πληροφορίες που χρειάζονται για να δημιουργηθεί ένας Growing Random γράφος. Το vertices είναι ο αριθμός των κορυφών που θα έχει ο γράφος και edges είναι ο αριθμός των ακμών που θα δημιουργηθούν στον γράφο.

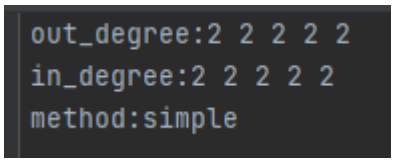
Σχήμα 4.8 Recent Degree input file

Στο σχήμα 4.8 είναι οι πληροφορίες που χρειάζονται για την δημιουργία ενός Recent Degree γράφου. Το vertices είναι ο αριθμός των κορυφών του γράφου, το edges είναι ο αριθμός των ακμών και το window είναι το μέγεθος του παραθύρου σε βήματα χρόνου.



Σχήμα 4.9 Static Fitness input file

Στο σχήμα 4.9 είναι οι πληροφορίες που χρειάζονται για να δημιουργηθεί ένα γράφημα Static Fitness. Το edges είναι ο αριθμός των ακμών που θα δημιουργηθούν στον γράφο, το fit_out είναι μία λίστα για το fitness_out αντίστοιχα για κάθε κορυφή και το fit_in είναι μία λίστα για το fitness_in αντίστοιχα για κάθε κορυφή.



Σχήμα 4.10 Degree Sequence input file

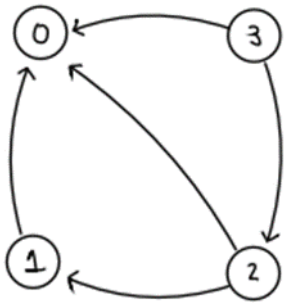
Στο σχήμα 4.10 βλέπουμε τις πληροφορίες που χρειάζεται το πρόγραμμα για την δημιουργία ενός Degree Sequence γράφου. Το out_degree είναι μία λίστα με την ακολουθία βαθμών για τις εξερχόμενες ακμές για όλες τις κορυφές αντίστοιχα, το in_degree είναι η ακολουθία βαθμού για τις εισερχόμενες ακμές αντίστοιχα για κάθε

κορυφή και το method είναι η μέθοδος που χρησιμοποιείται για την δημιουργία του γράφου.

Το πρόγραμμα στη συνέχεια δημιουργεί ένα γράφο ανάλογα με τι θα του πει ο χρήστης στο αρχείο input.txt στο κομμάτι του hypergraph. Όταν ο γράφος δημιουργηθεί το πρόγραμμα προχωρά στην δημιουργία των υπογράφων. Οι υπογράφοι θα είναι όσες και οι κορυφές του υπεργράφου που δημιουργήθηκε. Στη συνέχεια βάση του αριθμού των κορυφών του υπεργράφου δημιουργούνται οι αντίστοιχοι υπογράφοι. Το είδος του υπογράφου δίνεται από το αρχείο input.txt. Στη συνέχεια οι υπογράφοι που δημιουργήθηκαν ενώνονται με μία μέθοδο της βιβλιοθήκης python-igraph την union. Στη συνέχεια για κάθε ακμή που υπάρχει στον υπεργράφο που δημιουργήθηκε στην αρχή δημιουργούνται κάποιες ακμές μεταξύ των υπογράφων. Δηλαδή αν υπάρχει ακμή από την κορυφή 1 στην κορυφή 2 στον υπεργράφο τότε θα είναι ενωμένοι οι υπογράφοι 1 και 2 με κάποιων αριθμών από ακμές. Ο αριθμός αυτός μπορεί να είναι από 1 μέχρι και τον αριθμό που δίνει ο χρήστης στο αρχείο εισόδου input.txt. Ο αριθμός αυτός επιλέγεται τυχαία.

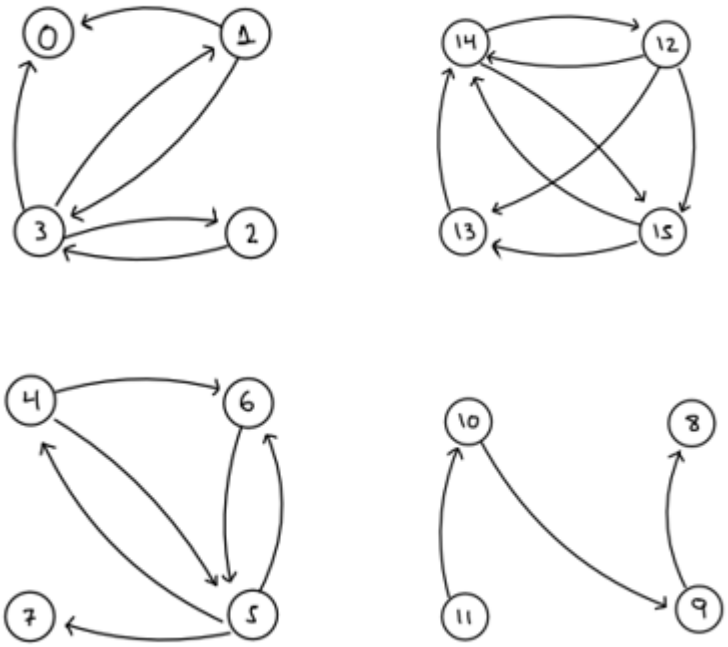
Στη συνέχεια όταν ολοκληρωθεί ο γράφος, το πρόγραμμα μετρά πόσες είναι οι μη-κατευθυνόμενες ακμές που έχουν δημιουργηθεί και υπολογίζει πόσες θα πρέπει να είναι μέσα στο πρόγραμμα βάση του ποσοστού που έδωσε ο χρήστης στο αρχείο input.txt.

Στο πιο κάτω παράδειγμα ο χρήστης επέλεξε να δημιουργήσει ένα Barabasi γράφο σαν τον υπεργράφο του τελικού γράφου με 4 κορυφές.



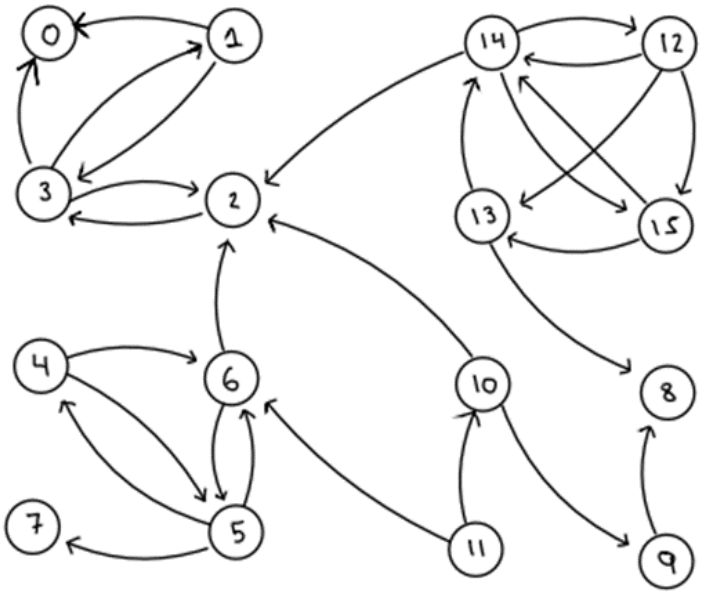
Σχήμα 4.11 Barabasi Graph

Στη συνέχεια ο χρήστης επέλεξε για υπογράφο του τελικού γράφου να είναι Erdos-Renyi με 4 κορυφές. Το πρόγραμμα δημιουργεί 4 διαφορετικούς υπογράφους αφού οι κορυφές του υπεργράφου που δημιουργήθηκε είναι τέσσερις.



Σχήμα 4.12 Erdos-Renyi subgraphs

Στο τελικό αποτέλεσμα θα υπάρχουν 4 υπογράφοι αφού υπεργράφος έχει 4 κορυφές. Ο τελικός γράφος φαίνεται στο σχήμα 4.13



Σχήμα 4.13 Τελικός γράφος του προγράμματος

Όπως βλέπουμε στο σχήμα 4.13 ο τελικός γράφος έχει 4 Erdos-Renyi γράφους. Για κάθε ακμή του υπεργράφου που δημιουργήσαμε στην αρχή βλέπουμε ότι υπάρχει μία ακμή από τα υπογραφήματα που ενώνονται στον υπεργράφο. Η κορυφή 0 του υπεργράφου είναι ο αντίστοιχος υπογράφος που έχει τις κορυφές 0-3 και αντίστοιχα οι υπόλοιποι. Στο παράδειγμα οι υπογράφοι ενώνονται μόνο με μία ακμή. Ο χρήστης μπορεί να δώσει ένα αριθμό ο οποίος θα είναι ο μέγιστος αριθμός ακμών μεταξύ διαφορετικών υπογραφημάτων. Ο αριθμός των ακμών μεταξύ διαφορετικών υπογραφημάτων επιλέγεται τυχαία από το 1 μέχρι και τον αριθμό που επέλεξε ο χρήστης. Οι κορυφές με τις οποίες θα ενώνονται οι υπογράφοι επίσης επιλέγονται τυχαία από το πρόγραμμα για να είναι ο τελικός γράφος όσο πιο τυχαίος γίνεται. Ο τελικός γράφος λοιπόν αποτελείται από 16 κορυφές.

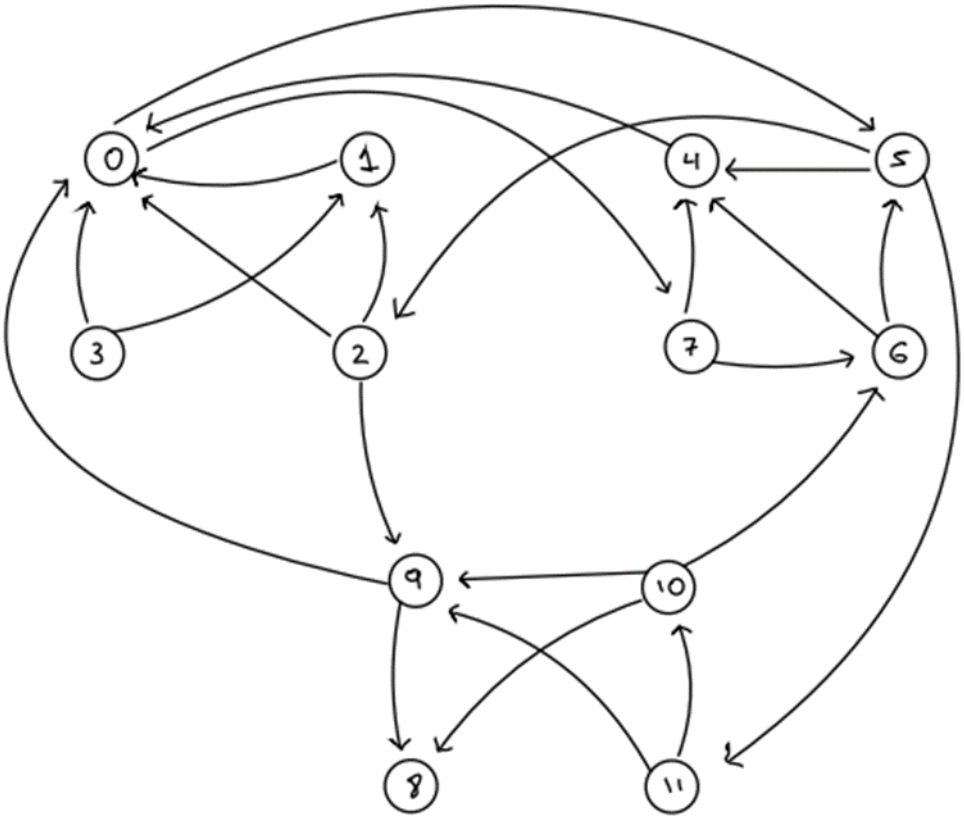
4.2 Ποσοστό μη-κατευθυνόμενων ακμών

Ο χρήστης επιλέγει το ποσοστό των μη-κατευθυνόμενων ακμών στο τελικό γράφο. Ανάλογα με τον αριθμό των ακμών και ποιο είναι το ποσοστό που επέλεξε ο χρήστης το πρόγραμμα θα αφαιρέσει ή θα προσθέσει ακμές για να φτάσει το θεμιτό ποσοστό μη-κατευθυνόμενων ακμών. Στο γράφο υπάρχουν κάποιοι περιορισμοί για το ποιες ακμές μπορούν να αφαιρεθούν ή να προστεθούν. Για να κρατηθεί το είδος του γράφου που επέλεξε ο χρήστης, κάθε φορά που αφαιρείται μία ακμή θα πρέπει να προστεθεί άλλη και όταν προστεθεί μία ακμή θα πρέπει να αφαιρεθεί μία άλλη.

Στην περίπτωση που θα πρέπει να αφαιρεθούν ακμές, δημιουργείται μία λίστα με όλες τις ακμές που είναι μη-κατευθυνόμενες στον γράφο. Από αυτή την λίστα επιλέγεται μία ακμή τυχαία από το πρόγραμμα και αποφασίζεται πάλι τυχαία ποια ακμή από τις δύο θα διαγραφεί. Μετά ελέγχετε από το πρόγραμμα αν μπορεί από την κορυφή που ξεκινά η ακμή που θα διαγραφεί, αν γίνεται να προστεθεί μία άλλη ακμή για να μην αλλάξει η πυκνότητα του γράφου. Οι ελέγχει που γίνονται είναι δύο, αν οι κορυφές που ενώνονται με αυτή την ακμή βρίσκονται στο ίδιο υπογράφημα τότε θα ελεγχθεί αν μπορεί να προστεθεί μία άλλη ακμή που να ξεκινά από την κορυφή που ξεκινά και η ακμή που θα διαγραφεί και να μην υπάρχει η αντίστροφη της. αν οι κορυφές βρίσκονται σε διαφορετικά υπογραφήματα τότε ελέγχεται αν γίνεται από αυτά τα υπογραφήματα να προστεθεί μία άλλη ακμή που να μην υπάρχει η αντίστροφή της.

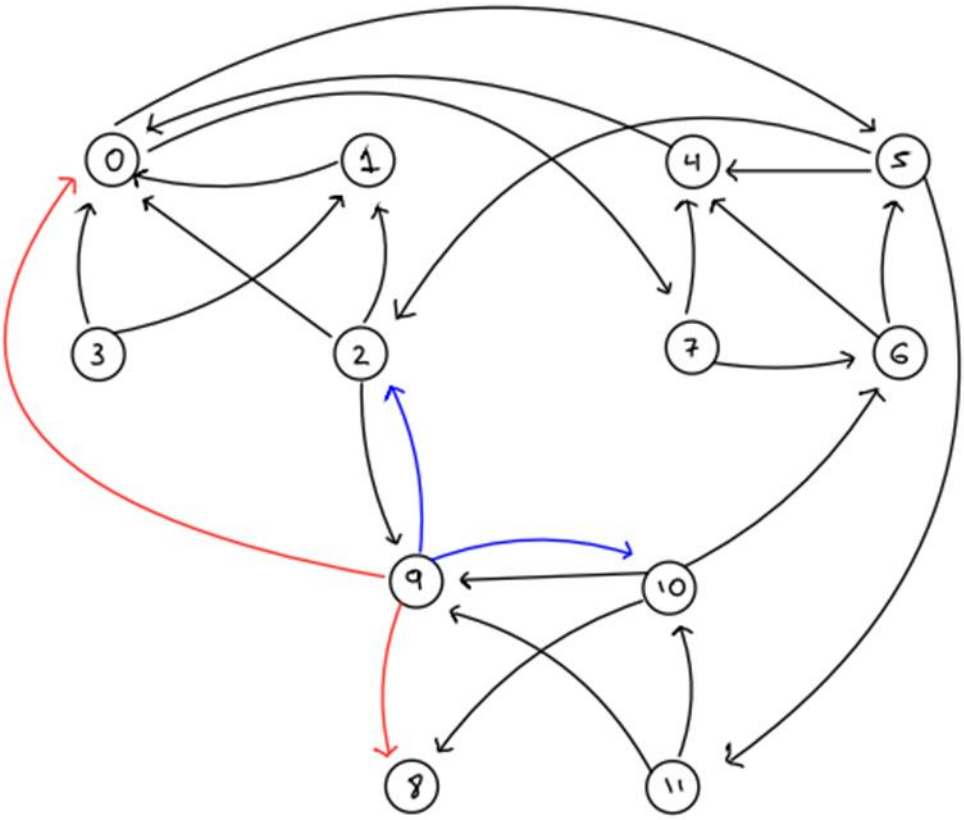
Στην περίπτωση που θα πρέπει να προστεθούν ακμές, δημιουργείται μία λίστα με όλες τις κατευθυνόμενες ακμές. Από αυτή τη λίστα επιλέγεται τυχαία μία ακμή και ελέγχεται αν γίνεται να προστεθεί η αντίστροφη της. Οι έλεγχοι είναι πάλι δύο, αν οι κορυφές που ενώνονται με αυτή την ακμή βρίσκονται στο ίδιο υπογράφημα τότε ελέγχεται αν γίνετε να διαγραφεί μία ακμή που ξεκινά από την κορυφή που είναι ο στόχος της ακμής που επιλέχθηκε, και αν οι κορυφές βρίσκονται σε διαφορετικά υπογραφήματα ελέγχεται αν γίνεται να προστεθεί ακμή από το υπογράφημα που είναι ο στόχος της ακμής που επιλέχθηκε προς το υπογράφημα που βρίσκεται η πηγή της ακμής. Ο έλεγχος που γίνετε εδώ είναι ότι αν προστεθεί η ακμή αυτή δεν θα ξεπερνά τον μέγιστο αριθμό ακμών μεταξύ διαφορετικών υπογραφημάτων που δήλωσε στην αρχή.

Παράδειγμα 1:



Σχήμα 4.14 Γράφος παραδείγματος 1

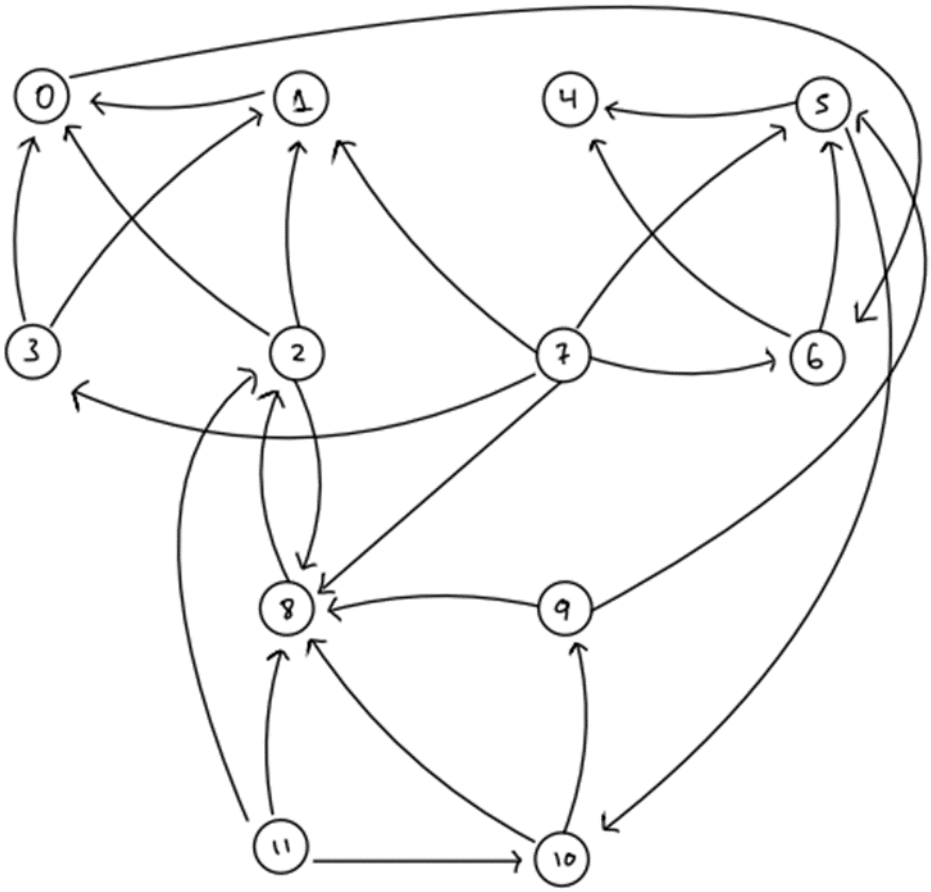
Στο παράδειγμα του σχήματος 4.14 ο χρήστης επέλεξε να δημιουργήσει k_regular γράφο σαν υπεργράφο και Barabasi γράφο σαν υπογράφο του τελικού γράφου. Ο k_regular έχει 3 κορυφές και οι Barabasi έχουν 4 κορυφές. Άρα ο τελικός γράφος έχει 12 κορυφές. Στο παράδειγμα αυτό το ποσοστό που έδωσε ο χρήστης για τις μη-κατευθυνόμενες ακμές είναι 10%. Στην αρχή ο γράφος δεν έχει καμία μη-κατευθυνόμενη ακμή όπως μπορούμε να δούμε στο σχήμα 4.14. Συνολικά στον γράφο υπάρχουν 23 ακμές. Το 10% είναι 2.3 άρα το πρόγραμμα θα πρέπει να προσθέσει 2 ακμές.



Σχήμα 4.15 Τελικός γράφος του παραδείγματος 1 με πρόσθεση ακμών

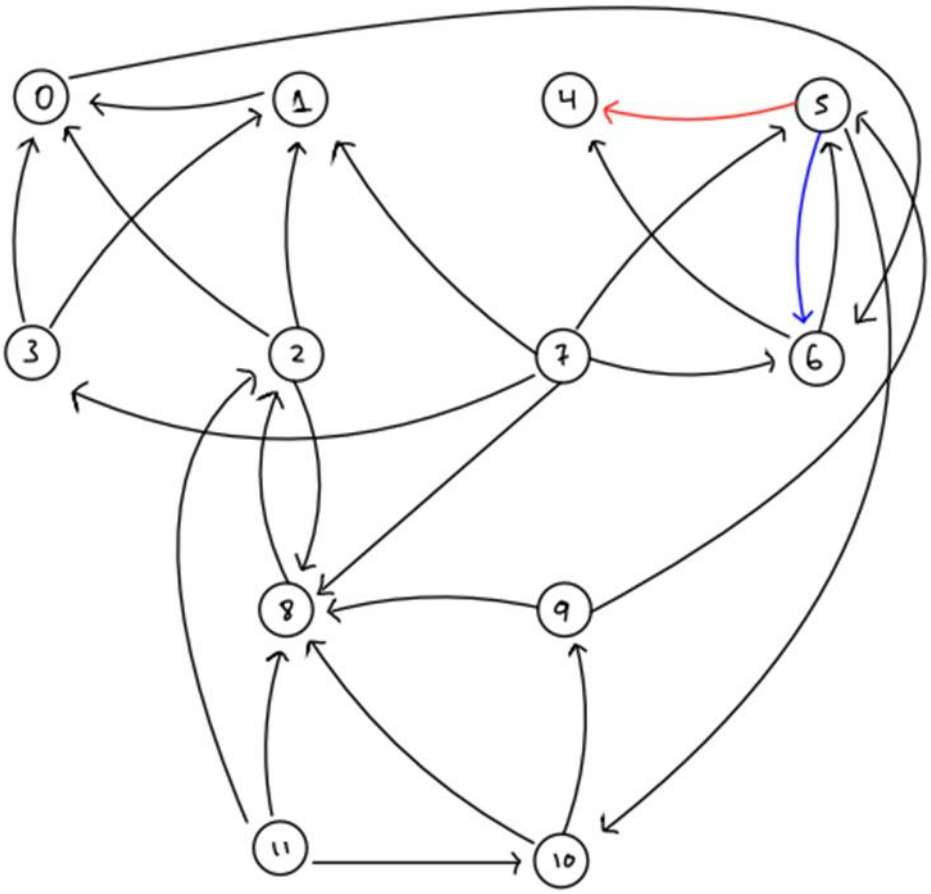
Οι ακμές με μπλε χρώμα είναι οι 2 ακμές που έχουν προστεθεί και οι ακμές με κόκκινο είναι οι ακμές που έπρεπε να διαγραφούν για να διατηρηθεί η ίδια πυκνότητα στον γράφο και να δημιουργηθούν 2 μη-κατευθυνόμενες ακμές. Δηλαδή υπήρχε η ακμή (2,9) και προστέθηκε η ακμή (9,2). Για να υπάρχει συνοχή με τον αρχικό γράφο αφού προστέθηκε μία ακμή από την κορυφή 9 προς την κορυφή 2 έπρεπε να διαγραφεί άλλη ακμή από την ίδια κορυφή προς κάποια κορυφή στον ίδιο υπογράφο με την κορυφή 2. Στον τελικό γράφο οι 2 μη κατευθυνόμενες ακμές είναι οι (2,9) και (9,10).

Παράδειγμα 2:



Σχήμα 4.16 Τελικός γράφος παραδείγματος 2

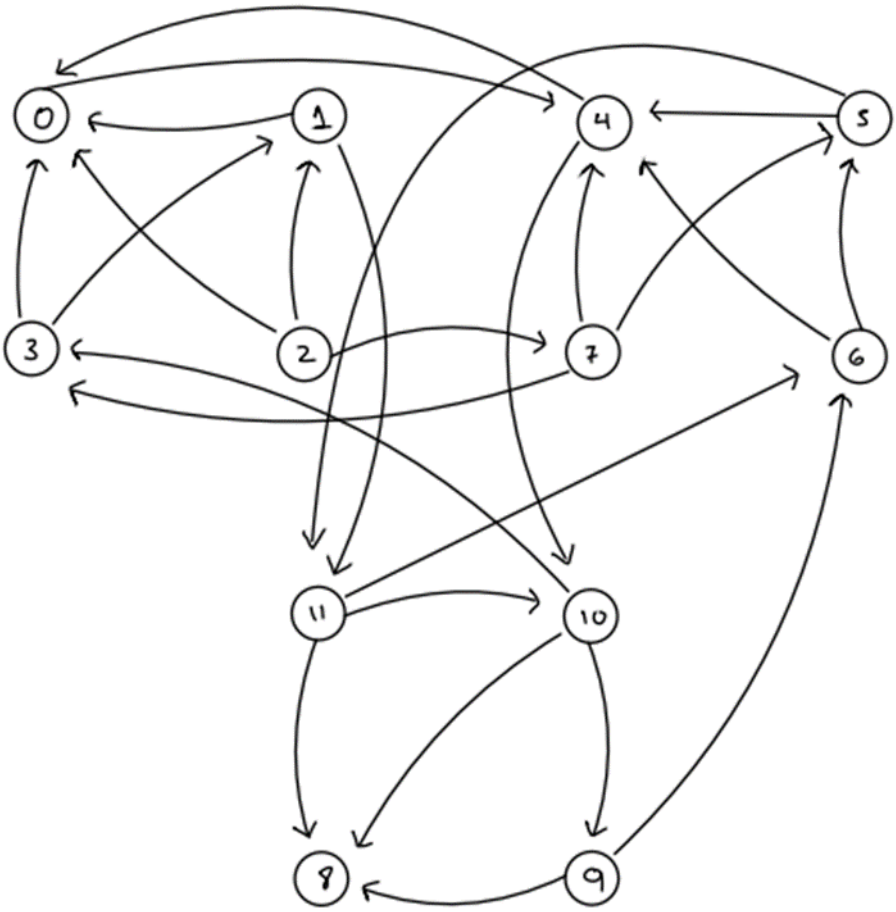
Στο σχήμα 4.16 ο χρήστης επέλεξε ακριβώς τα ίδια χαρακτηριστικά των γράφων με το παράδειγμα 1. Σε αυτό το παράδειγμα ο αρχικός αριθμός μη-κατευθυνόμενων ακμών είναι 1. Η μη κατευθυνόμενη ακμή είναι η (2,8). Ο συνολικός αριθμός ακμών στον γράφο είναι 24 και ο χρήστης επέλεξε 10% να είναι μη κατευθυνόμενες. Άρα στον τελικό γράφο θα πρέπει να υπάρχουν 2 μη κατευθυνόμενες ακμές. Στο τελικό γράφο θα προστεθεί 1 ακμή.



Σχήμα 4.17 Τελικός γράφος του παραδείγματος 2 μετά την προσθήκη μίας ακμής

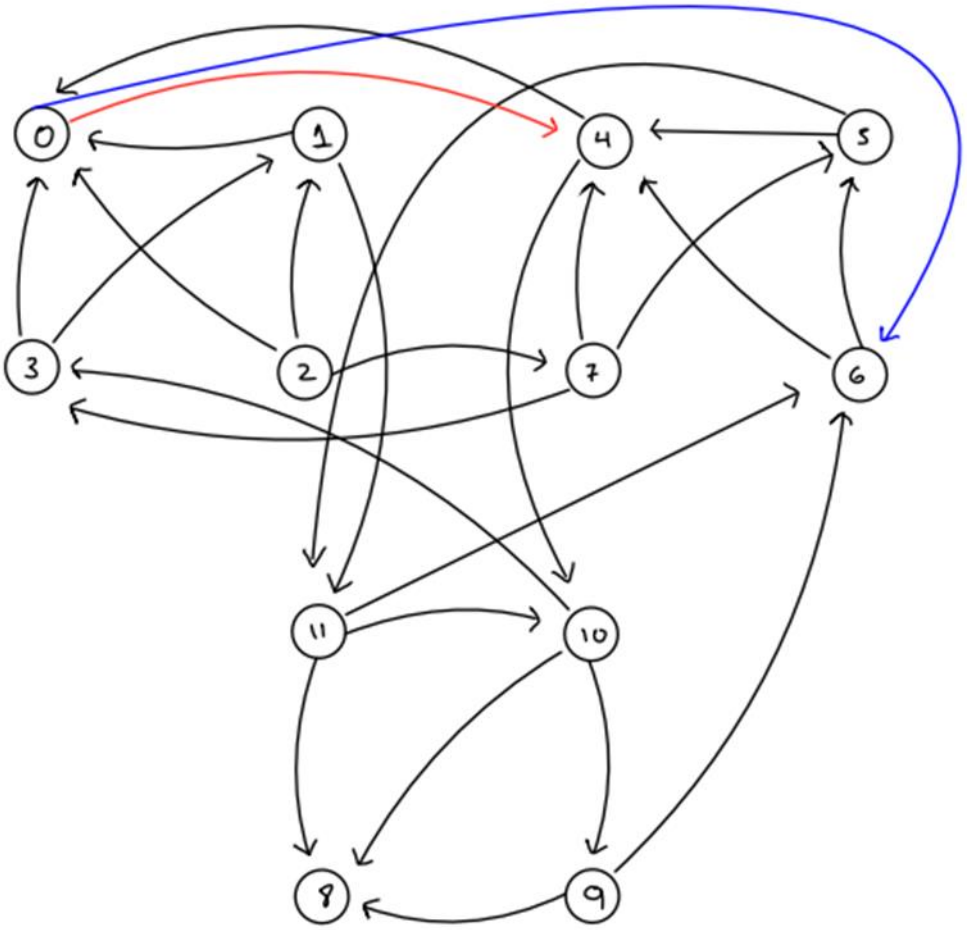
Με μπλε είναι η ακμή που έχει προστεθεί και με κόκκινο η ακμή που έχει αφαιρεθεί.

Παράδειγμα 3:



Σχήμα 4.18 Τελικός γράφος παραδείγματος 3

Στο παράδειγμα του σχήματος 4.18 τα χαρακτηριστικά των γράφων είναι τα ίδιο. Το μόνο που άλλαξε εδώ είναι το ποσοστό των μη-κατευθυνόμενων ακμών. Από 10% που ήταν στα προηγούμενα παραδείγματα τώρα είναι 0%. Άρα στον τελικό γράφο δεν τα υπάρχει καμία κατευθυνόμενη ακμή. Όπως βλέπουμε στο πιο πάνω γράφο υπάρχει μία μη κατευθυνόμενη ακμή η (0,4). Αφού το ποσοστό είναι 0% τότε πρέπει να διαγραφεί μία από τις ακμές (0,4) ή (4,0) για να μην είναι μη κατευθυνόμενη.



Σχήμα 4.19 Τελικός γράφος του παραδείγματος 3 μετά την αφαίρεση της μίας ακμής

Το πρόγραμμα επέλεξε τυχαία μία από τις 2 ακμές που θα διαγραφούν. Έχει διαγραφεί η ακμή (0,4) και έχει προστεθεί η ακμή (0,6).

Κεφάλαιο 5

Πειράματα

5.1 Εισαγωγή

Σκοπός της διπλωματικής μου εργασίας ήταν να δημιουργήσω θεωρίες επιχειρηματολογίας στη μορφή ενός γράφου με τον τρόπο που εξηγήθηκε πιο πάνω για να χρησιμοποιηθούν σε κάποια πειράματα με σκοπό να εξάγουμε κάποια συμπεράσματα σχετικά με τις ιδιότητες των θεωριών αυτών με βάση τα δομικά τους χαρακτηριστικά. Στα πειράματα αυτά χρησιμοποιήθηκαν διάφορα είδη τυχαίων γραφημάτων από την λίστα που ανέφερα σε προηγούμενο κεφάλαιο με αποτέλεσμα να έχουμε μία πιο ολοκληρωμένη άποψη για διάφορα είδη γράφων.

Οι θεωρίες που δημιουργήθηκαν δίνονταν σαν είσοδο σε ένα πρόγραμμα συνόλου απαντήσεων. Το αποτέλεσμα του προγράμματος αυτού είναι ο αριθμός των ευσταθών μοντέλων που εντοπίστηκαν στην θεωρία που δίνεται ως είσοδο και ο χρόνος που χρειάστηκε για να εντοπιστεί ο αριθμός αυτός. Στο πρόγραμμα δίνεται σαν ανώτατο όριο ευσταθών μοντέλων που θα εντοπίσει ο αριθμός 100, αυτό γίνεται για να τελειώσει το πρόγραμμα σύντομα και όσες θεωρίες έχουν 100 μοντέλα σημαίνει ότι έχουν πολύ περισσότερα αλλά δεν μας απασχολεί ακριβώς ο αριθμός τους όταν είναι περισσότερα από 100.

5.2 Συνδυασμοί γραφημάτων

Για να έχουμε μία πιο ολοκληρωμένη άποψη για τα πειράματα χρειάστηκε να τρέξω το πρόγραμμα με διάφορα είδη γραφημάτων και διάφορες παράμετρούς.

Για κάθε συνδυασμό γραφημάτων δινόταν ένα είδος για υπεργράφο και ένα είδος για υπογράφο. Οι συνδυασμοί που μελετήθηκαν είναι (Υπεργράφος/Υπογράφος):

1. Barabasi/Barabasi
2. Erdos/Erdos
3. Barabasi/Erdos
4. Erdos/Barabasi
5. Static Power Law/Erdos
6. Erdos/Static Power Law
7. K_Regular/Erdos
8. K_Regular/Barabasi

Για όλα τα πειράματα ο αριθμός τω κορυφών του υπεργράφου ήταν 5 και ο αριθμός των κορυφών των υπογράφων ήταν 200, άρα οι γράφοι που δημιουργούνται περιέχουν 1000 κορυφές. Η πυκνότητα των γράφων που δημιουργούνταν ήταν 0.05.

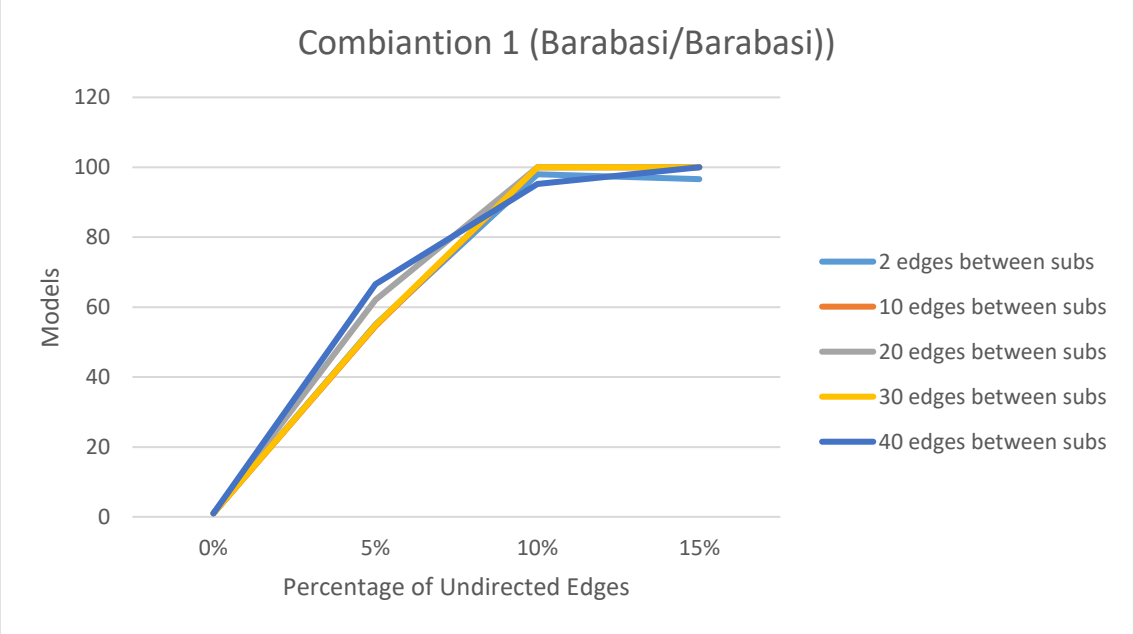
Επιπλέον για τα πειράματα δόθηκαν διάφορες τιμές για τον αριθμό των ακμών μεταξύ των υπογραφημάτων και διάφορα ποσοστά για τις μη-κατευθυνόμενες ακμές που θα υπάρχουν στο γράφο. Για κάθε συνδυασμό γραφήματος δημιουργήθηκαν 20 τυχαία γραφήματα για κάθε αριθμό ενδιάμεσων ακμών μεταξύ υπογράφων κα για κάθε ποσοστό μη-κατευθυνόμενων ακμών.

Οι αριθμοί των ακμών μεταξύ των υπογράφων ήταν 2, 10, 20, 30, 40. Για κάθε αριθμό από αυτούς δημιουργήθηκαν 20 γραφήματα με 0% μη-κατευθυνόμενες ακμές, 20 γραφήματα με 5% μη-κατευθυνόμενες ακμές, 20 γραφήματα με 10% μη-κατευθυνόμενες ακμές και 20 γραφήματα με 15% μη-κατευθυνόμενες ακμές. Άρα σύνολο για κάθε αριθμό ακμών μεταξύ των υπογράφων δημιουργήθηκαν 80 τυχαία γραφήματα.

Το κάθε γράφημα από αυτά αντιπροσωπεύει μία θεωρία επιχειρηματολογίας. Αυτή η θεωρία γράφτηκε σε ένα αρχείο theory.lp το οποίο δινόταν σαν είσοδο στο αρχείο stable.lp το οποίο εντοπίζει ευσταθεί μοντέλα μέσα στην θεωρία. Για το κάθε παράδειγμα που έτρεξα σημείωσα τα μοντέλα που εντοπίστηκαν και τον χρόνο ου έκανε το πρόγραμμα για να τα εντοπίσει.

5.3 Αποτελέσματα

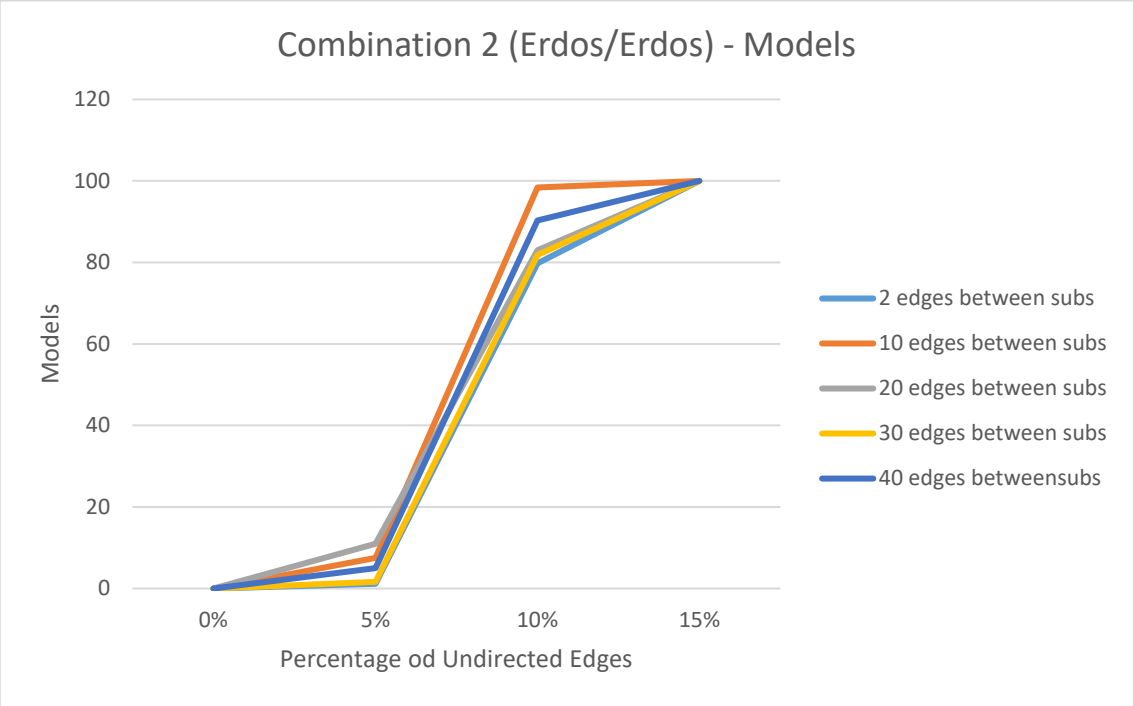
Όλοι οι γράφοι που δημιουργήθηκαν είναι με πυκνότητα 5%. Για το πρώτο συνδυασμό γραφημάτων στην αρχή δημιουργήθηκαν 80 γράφοι με 2 ακμές μεταξύ των υπογράφων όπου 20 από αυτούς ήταν με 0% ποσοστό μη-κατευθυνόμενων ακμών, 20 γράφοι με ποσοστό 5% μη-κατευθυνόμενων ακμών, 20 γράφοι με ποσοστό 10% μη-κατευθυνόμενων ακμών και τέλος ακόμη 20 με ποσοστό 15%. Με τον ίδιο τρόπο συνεχίστηκαν τα πειράματα, για κάθε αριθμό ακμών μεταξύ των υπογράφων δημιουργούνταν 80 γράφοι για τα 4 διαφορετικά ποσοστά μη-κατευθυνόμενων ακμών.



Σχήμα 5.1 Μοντέλα 1ου συνδυασμού με πυκνότητα 5%

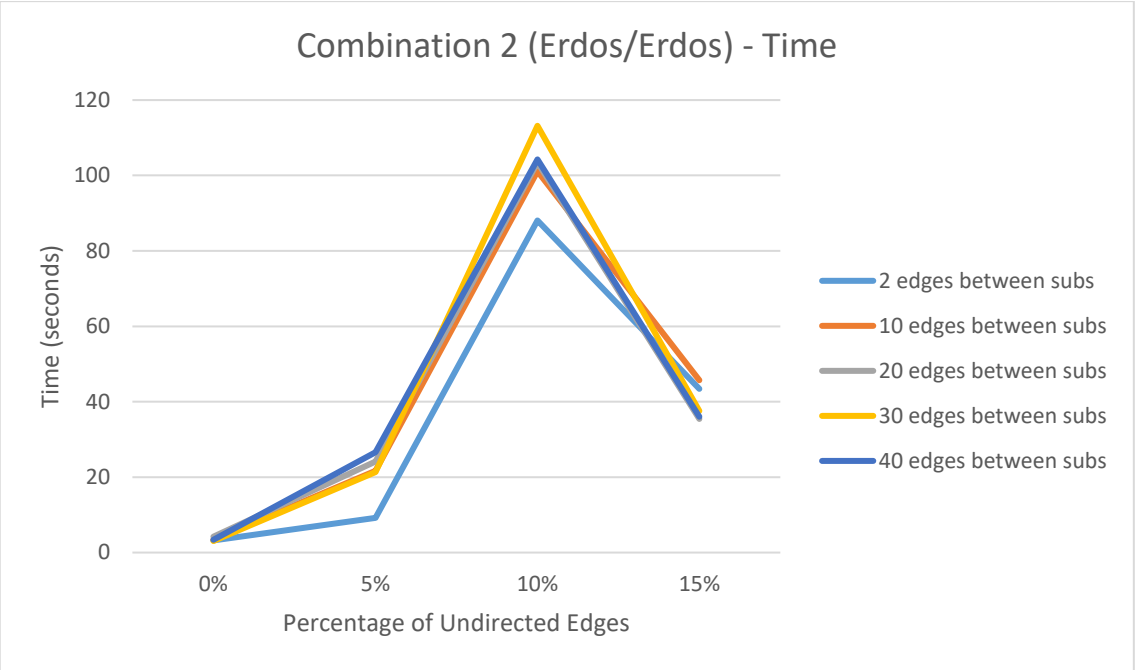
Στη γραφική παράσταση του σχήματος 5.1 παρατηρούμε την συμπεριφορά των μοντέλων που εντοπίστηκαν στους διάφορους γράφους. Με ποσοστό 0% μη-κατευθυνόμενων ακμών παρατηρούμε ότι και στις 5 περιπτώσεις διαφορετικών αριθμών των ακμών μεταξύ των υπογράφων ο αριθμός των μοντέλο είναι κοντά στο 0, για την ακρίβεια

παντού είναι 1. Στη συνέχεια όσο αυξάνεται το ποσοστό μη-κατευθυνόμενων ακμών αυξάνεται και ο αριθμός των μοντέλων που εντοπίζονται στους γράφους. Σημειώνεται και ότι, σε κάθε ποσοστό μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων που φαίνεται στην γραφική είναι ο μέσος όρος των 20 τυχαίων γραφημάτων που δημιουργήθηκαν.



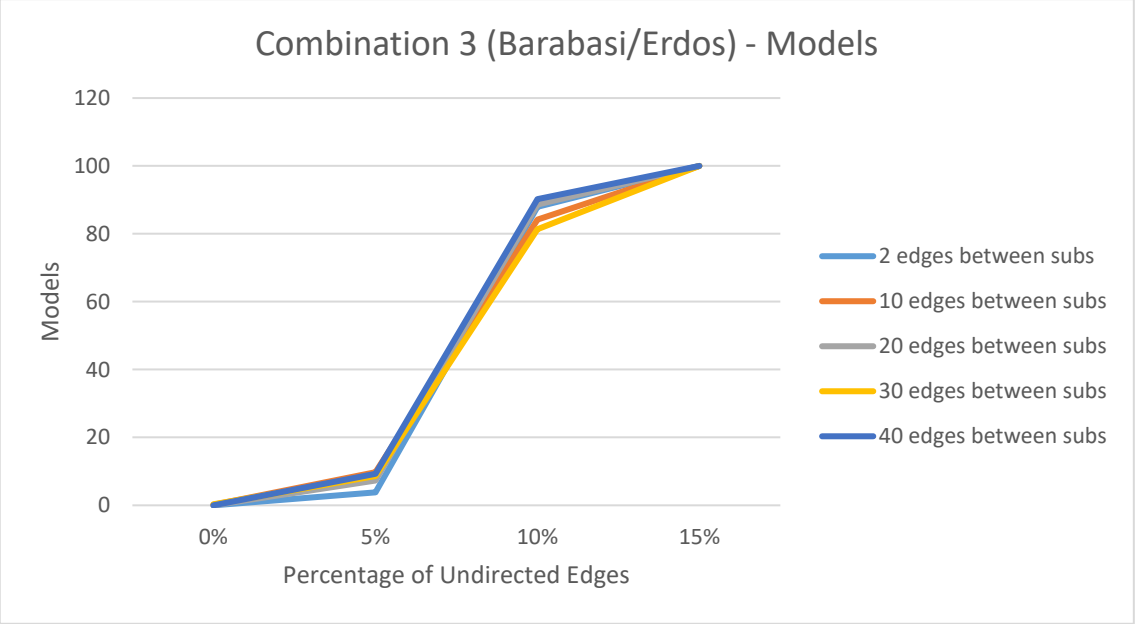
Σχήμα 5.2 Μοντέλα 2ου συνδυασμού

Στο σχήμα 5.2 βλέπουμε τον μέσο όρο των μοντέλων που εντοπίστηκαν στα 20 τυχαία γραφήματα του 2ου συνδυασμού. Παρατηρούμε και εδώ όπως και στον 1ο συνδυασμό ότι τα μοντέλα όσο αυξάνεται το ποσοστό μη κατευθυνόμενων ακμών στο γράφημα αυξάνεται και ο αριθμός των μοντέλων που εντοπίστηκαν. Όμως σε αυτό τον συνδυασμό παρατηρούμε ότι σε ποσοστά 0% και 5% μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων παραμένει σε χαμηλά επίπεδα και αυξάνεται απότομα ο αριθμός των μ0οντέλων όταν το ποσοστό μη-κατευθυνόμενων ακμών γίνει 10% και παραμένει σε υψηλά επίπεδα και στο 15% μη-κατευθυνόμενων ακμών.



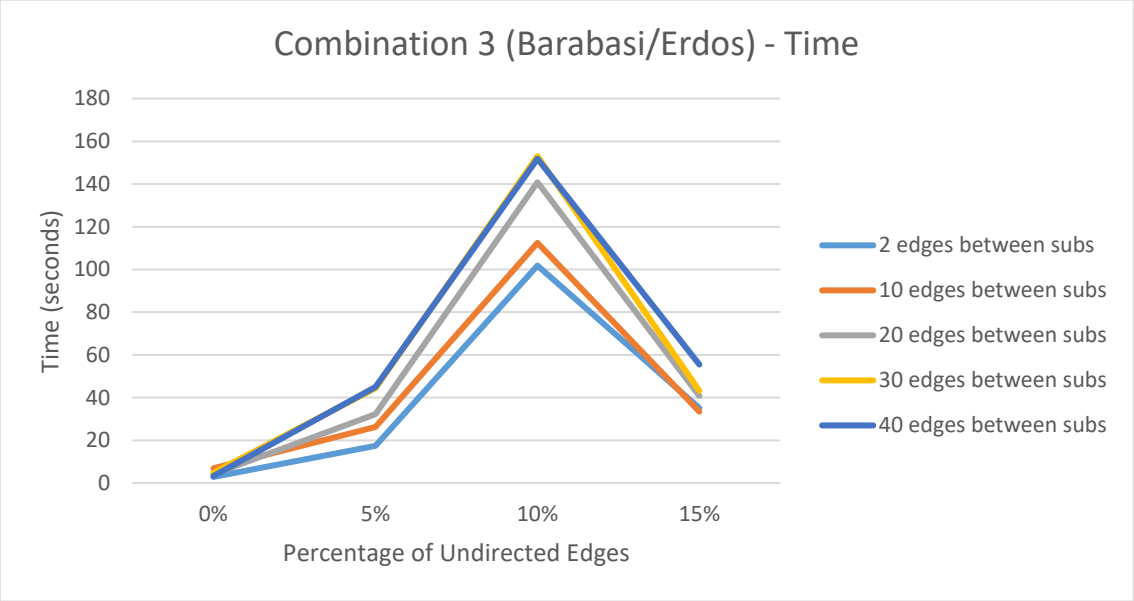
Σχήμα 5.3 Χρόνος 2ου συνδυασμού

Στο σχήμα 5.3 βλέπουμε το χρόνο σε δευτερόλεπτα που χρειάστηκε το πρόγραμμα για να εντοπίσει τον αριθμό των μοντέλων που φαίνονται στο σχήμα 5.2. Παρατηρούμε ότι μέχρι το 10% μη-κατευθυνόμενων ακμών ο χρόνος που χρειάζεται το πρόγραμμα αυξάνεται άλλα στο 15% βλέπουμε ότι ο χρόνος που χρειάζεται για να τα εντοπίσει μειώνεται περίπου στο μισό σε σχέση με το 10% μη κατευθυνόμενων ακμών.



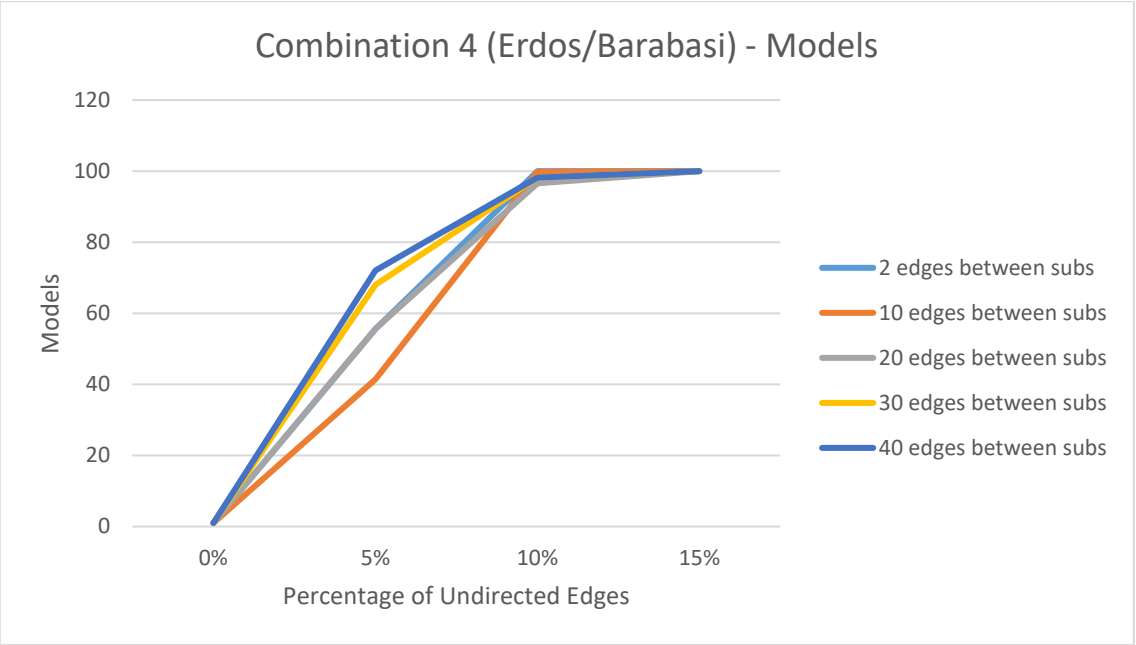
Σχήμα 5.4 Μοντέλα 3ου συνδυασμού

Στο σχήμα 5.4 βλέπουμε τον αριθμό των μοντέλων που εντοπίστηκαν στα 20 γραφήματα που δημιουργήθηκαν για τον 3ο συνδυασμό γραφημάτων. Παρατηρούμε ότι όπως και στον 2ο συνδυασμό τα ποσοστά 0% και 5% μη-κατευθυνόμενων ακμών έχουν σχεδόν τον ίδιο αριθμό μοντέλων και παρατηρείται μία τεράστια αύξηση του αριθμού αυτού μετά το 10% των μη-κατευθυνόμενων ακμών στο γράφο. Στη συνέχεια και στο 15% των μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων είναι 100 σε όλους τους αριθμούς των ενδιάμεσων ακμών στους υπογράφους.



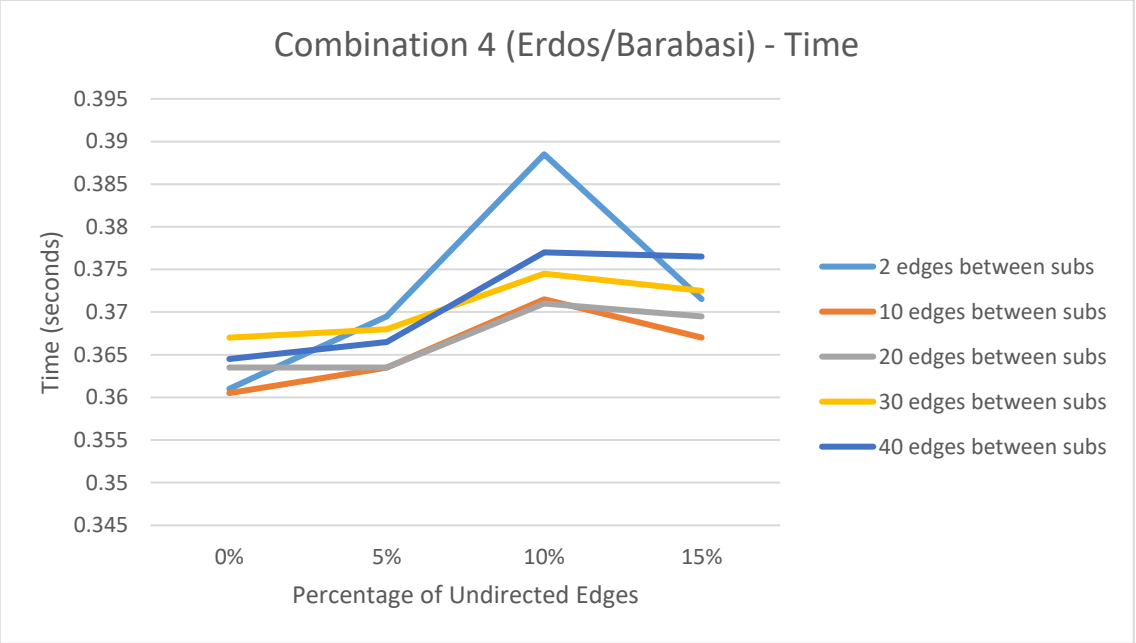
Σχήμα 5.5 Χρόνος 3ου συνδυασμού

Στο σχήμα 5.5 βλέπουμε τον χρόνο που χρειάστηκε το πρόγραμμα για να εντοπίσει τον αριθμό των μοντέλων. Βλέπουμε και εδώ ότι ο χρόνος αυξάνεται μέχρι το ποσοστό 10% των μη-κατευθυνόμενων ακμών αλλά μετά στο 15% των μη-κατευθυνόμενων ακμών μειώνεται αρκετά. Στον 3ο συνδυασμό βλέπουμε ότι ο μέγιστος αριθμός σε δευτερόλεπτα είναι περίπου 150 δευτερόλεπτα ενώ στον 2ο συνδυασμό ο μέγιστος αριθμός ήταν περίπου 110. Μέχρι το 10% το πρόγραμμα για τον συνδυασμό 3 χρειαζόταν περισσότερο χρόνο σε σχέση με τον χρόνο που χρειαζόταν το πρόγραμμα για τον συνδυασμό 2, όμως για το ποσοστό 15% το πρόγραμμα και για τους δύο συνδυασμούς ήθελε περίπου τον ίδιο χρόνο.



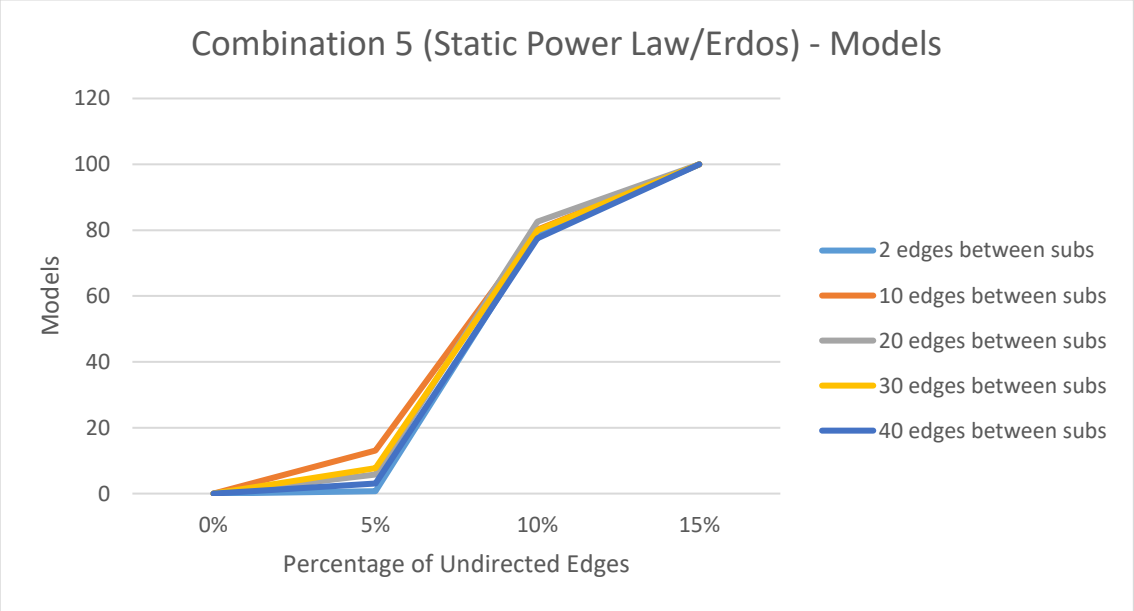
Σχήμα 5.6 Μοντέλα 4ου συνδυασμού

Στο σχήμα 5.6 βλέπουμε τον μέσο όρο των μοντέλων που εντόπισε το πρόγραμμα στα 20 τυχαία γραφήματα του 4ου συνδυασμού. Εδώ βλέπουμε κοινά με τον 1ο συνδυασμό στον οποίο από το 0% μέχρι και το 10% ο αριθμός των μοντέλων αυξανόταν συνεχώς και στο 15% βλέπουμε ότι παραμένει στο πιο ψηλό επίπεδο.



Σχήμα 5.7 Χρόνος 4ου συνδυασμού

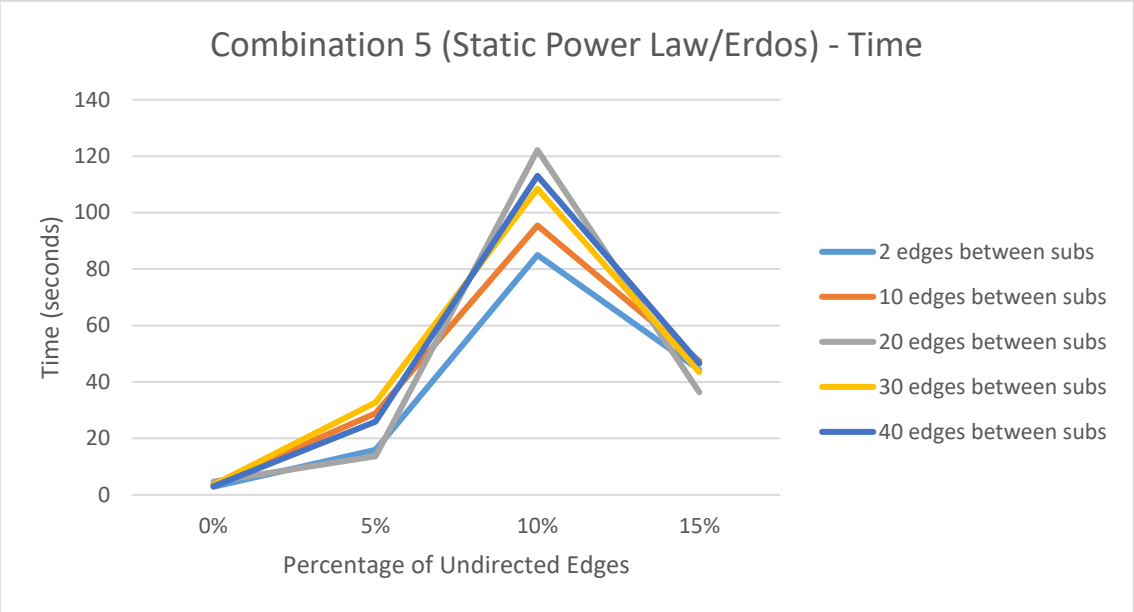
Στο σχήμα 5.7 βλέπουμε το χρόνο που χρειάστηκε το πρόγραμμα για να εντοπίσει τα μοντέλα. Στο πρώτο συνδυασμό δεν είδαμε γραφική με τους χρόνους επειδή για όλα τα παραδείγματα το πρόγραμμα ήθελε περίπου 0.4 δευτερόλεπτα. Εδώ βλέπουμε ακριβώς το ίδιο με τον 1ο συνδυασμό και εδώ το πρόγραμμα χρειάστηκε περίπου κατά μέσο όρο 0.37 δευτερόλεπτα για να εντοπίσει τα μοντέλα. Παρατηρούμε ότι ο συνδυασμός 1 και 4 έχουν και παρόμοιο αριθμό μοντέλων και παρόμοιο χρόνο και αυτό οφείλεται στο ότι και στους δύο συνδυασμούς το είδος του υπογράφου ήταν Barabasi. Μπορούμε επίσης να παρατηρήσουμε την διαφορά μεταξύ των συνδυασμών 1 και 4 με τους συνδυασμούς 2 και 3. Βλέπουμε ότι στους συνδυασμούς 2 και 3 ο χρόνος που χρειάστηκε ήταν περίπου 200 φορές περισσότερος από τον χρόνο που χρειάστηκε για να εντοπιστούν τα μοντέλα στους συνδυασμούς 1 και 4.



Σχήμα 5.8 Μοντέλα 5ου συνδυασμού

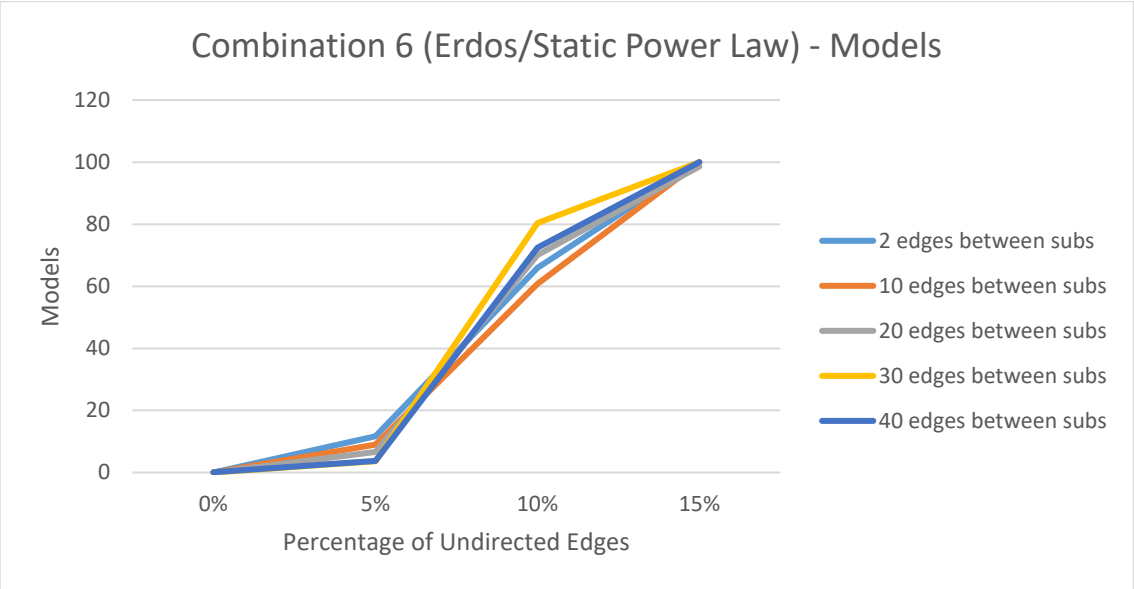
Στο σχήμα 5.8 βλέπουμε τον μέσο όρο των μοντέλων που εντοπίστηκαν στα πειράματα που έγιναν με τον 5ο συνδυασμό. Παρατηρούμε ξανά ότι για ποσοστό μη-κατευθυνόμενων ακμών 0% και 5% ο αριθμός των μοντέλων είναι αρκετά χαμηλός, κοντά στο 0 και παρατηρούμε ξανά μία απότομη αύξηση του αριθμού των μοντέλων που εντοπίστηκαν στο 10% των μη-κατευθυνόμενων ακμών. Σε αυτό τον συνδυασμό παρατηρείται ότι στο 10% μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων που εντοπίστηκαν είναι περίπου 80 για όλα τα πειράματα με όλους τους αριθμός ενδιάμεσων ακμών μεταξύ των υπογράφων. Κάτι παρόμοιο παρατηρήθηκε στον 2ο συνδυασμό και στον 3ο.Αυτό φαίνεται να επηρεάζεται από το είδος του υπογράφου στις 3 περιπτώσεις

αυτές. Στη συνέχεια όταν το ποσοστό μη-κατευθυνόμενων ακμών έγινε 15% βλέπουμε ότι για όλα τα παραδείγματα ο αριθμό των μοντέλων που εντοπίστηκαν είναι 100, όπως ακριβώς και στους συνδυασμούς 2 και 3 που παρατηρήθηκαν κάποια κοινά.



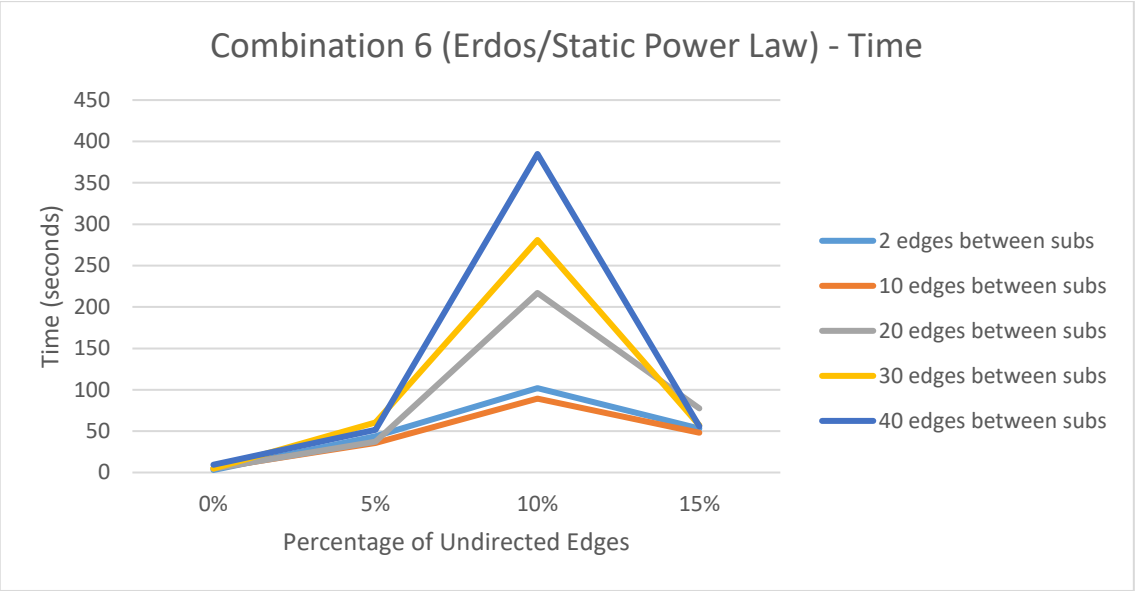
Σχήμα 5.9 Χρόνος 5ου συνδυασμού

Στο σχήμα 5.9 βλέπουμε το χρόνο που χρειάστηκε το πρόγραμμα για να εντοπίσει τα μοντέλα. Αρχικά βλέπουμε για όλα τα παραδείγματα μια αύξηση του χρόνου από το ποσοστό 0% μέχρι και το 10 % όπως παρατηρήθηκε ξανά σε προηγούμενους συνδυασμούς. Στους γράφους με 10% ποσοστό μη-κατευθυνόμενων ακμών παρατηρείται αρκετά μεγάλη διαφορά στον μέσο χρόνο που χρειάστηκε το πρόγραμμα για να εντοπίσει τα μοντέλα στα παραδείγματα με 20, 30 και 40 ενδιάμεσες ακμές μεταξύ των υπογράφων σε σχέση με τον χρόνο που χρειάστηκε για τα παραδείγματα με 2 και 10 ενδιάμεσες ακμές. Βλέπουμε ότι χρειάστηκε περίπου 120 δευτερόλεπτα για να εντοπίσει το πρόγραμμα τα μοντέλα για τα παραδείγματα με 20 ενδιάμεσες ακμές ενώ για 2 ενδιάμεσες ακμές το πρόγραμμα χρειάστηκε περίπου 85 δευτερόλεπτα. Τέλος, όπως και στο 2ο και στον 3ο συνδυασμό το πρόγραμμα για 15% ποσοστό μη-κατευθυνόμενων ακμών παρατηρήθηκε ότι ο χρόνος που χρειάστηκε ήταν περίπου ο μισός σε σχέση με τον χρόνο που χρειάστηκε για να εντοπίσει τα μοντέλα στα παραδείγματα με το 10% ποσοστό μη-κατευθυνόμενων ακμών.



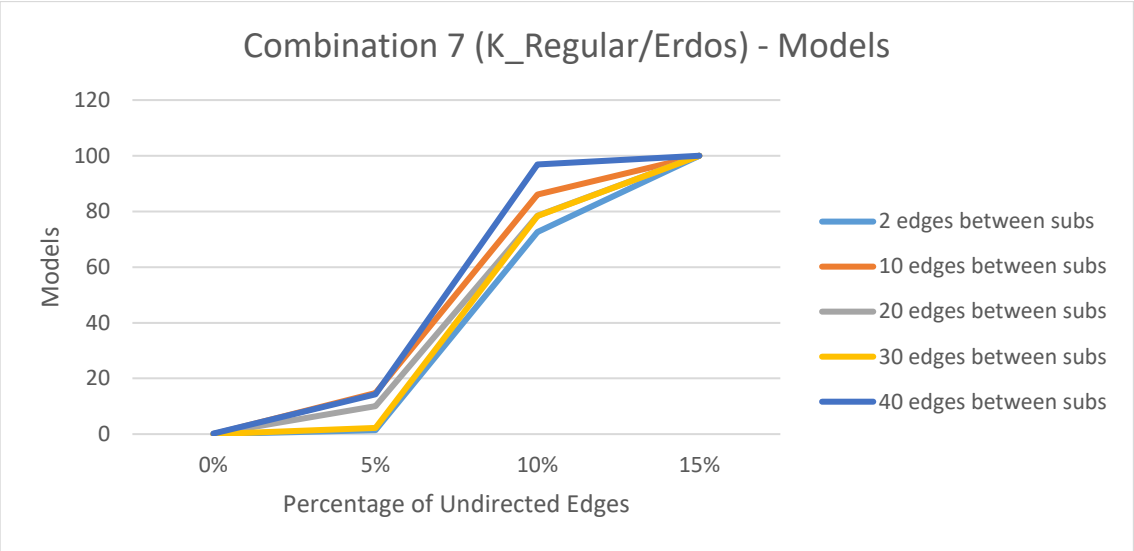
Σχήμα 5.10 Μοντέλα 6ου συνδυασμού

Στο σχήμα 5.10 βλέπουμε τον μέσο αριθμό των μοντέλων που εντοπίστηκαν στα γραφήματα του 6ου συνδυασμού. Παρατηρούμε όπως και στα υπόλοιπα ότι όσο αυξάνεται το ποσοστό μη-κατευθυνόμενων ακμών αυξάνεται και ο αριθμός των μοντέλων που εντοπίζονται. Σε αυτό το συνδυασμό παρατηρούμε ότι στην αρχή με ποσοστό 0% μη-κατευθυνόμενων ακμών τα μοντέλα που εντοπίζονται είναι σε όλα τα παραδείγματα 0. Στη συνέχεια σε ποσοστό 5% μη-κατευθυνόμενων ακμών βλέπουμε ότι πάλι σε όλα τα παραδείγματα ο αριθμός είναι αρκετά χαμηλός με τον πιο ψηλό να είναι περίπου 10. Ωστόσο, παρατηρούμε μία τεράστια αύξηση στο 10% μη-κατευθυνόμενων ακμών, που φτάνει περίπου 70 μοντέλα. Ο πιο χαμηλός αριθμός μοντέλων που εντοπίζονται με ποσοστό 10% είναι περίπου 60 και ο πιο ψηλός είναι περίπου 80 με 10 ενδιάμεσες ακμές μεταξύ των υπογράφων και με 30 ενδιάμεσες ακμές αντίστοιχα. Η διαφορά αυτή είναι σημαντική αλλά δεν υπάρχει κάποιος ιδιαίτερος λόγος που συμβαίνει αυτό. Ο μόνος λόγος που μπορεί να το εξηγήσει αυτό είναι η τυχαιότητα παραγωγής των γραφημάτων. Τέλος σε ποσοστό 15% μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων που εντοπίστηκαν είναι 100 σε όλα τα παραδείγματα.



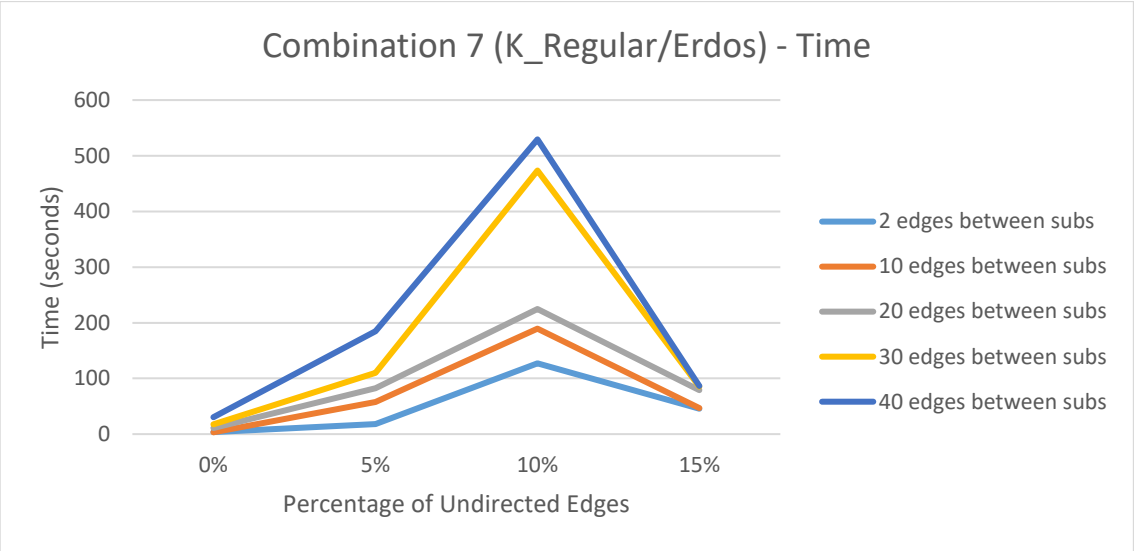
Σχήμα 5.11 Χρόνος 6ου συνδυασμού

Στο σχήμα 5.11 βλέπουμε τον μέσο χρόνο που χρειάστηκε το πρόγραμμα για να εντοπίσει τα μοντέλα. Παρατηρούμε στην αρχή για ποσοστό 0% μη-κατευθυνόμενων ακμών ο χρόνος είναι πολύ μικρός σχεδόν στο 0. Στο 5% μη-κατευθυνόμενων ακμών ο χρόνος πηγαίνει περίπου στα 50 δευτερόλεπτα σε όλα τα παραδείγματα. Όμως στο ποσοστό 10% μη-κατευθυνόμενων ακμών βλέπουμε ότι για τα παραδείγματα με 2 και 10 ενδιάμεσες ακμές ο χρόνος είναι περίπου στα 100 δευτερόλεπτα και στα υπόλοιπα 3 ο χρόνος αυξάνεται υπερβολικά. Στα παραδείγματα με 20 ενδιάμεσες ακμές ο χρόνος είναι περίπου 210 δευτερόλεπτα, στα παραδείγματα με 30 ενδιάμεσες ακμές είναι περίπου στα 275 δευτερόλεπτα και στα παραδείγματα με 40 ενδιάμεσες ακμές στους υπογράφους χρειάζεται περίπου 380 δευτερόλεπτα.



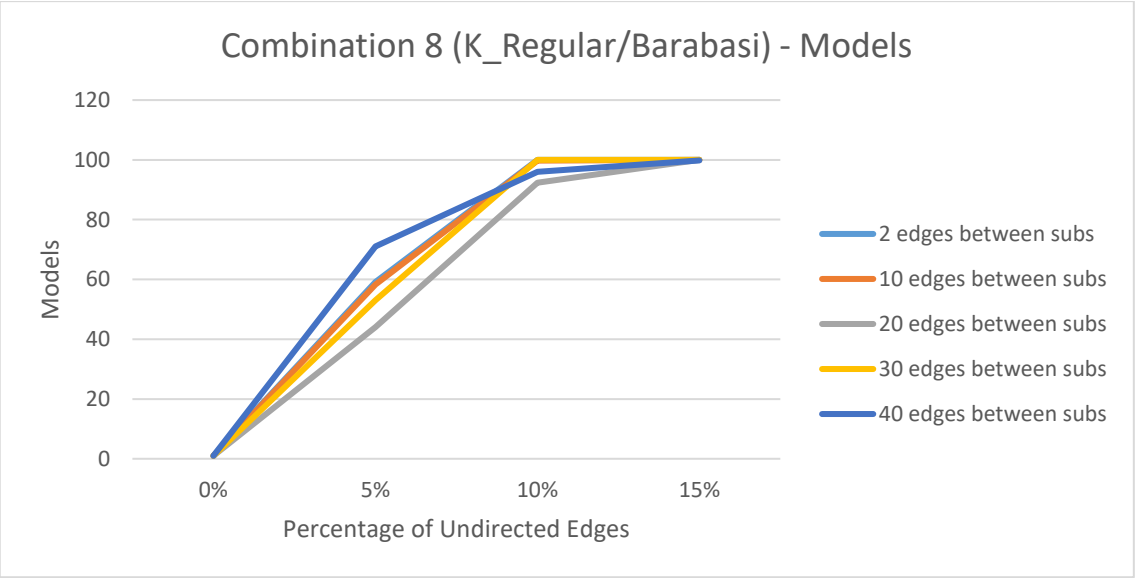
Σχήμα 5.12 Μοντέλα 7ου συνδυασμού

Στο σχήμα 5.12 βλέπουμε τον αριθμό των μοντέλων που εντοπίστηκαν για τον 7ο συνδυασμό γραφημάτων. Παρατηρούμε όπως και στα προηγούμενα παραδείγματα ότι ο αριθμός των μοντέλων που εντοπίστηκαν για τα παραδείγματα με ποσοστό 0% και 5% μη-κατευθυνόμενων ακμών είναι σχετικά μικρός και παρατηρείται μία αύξηση πάλι στο ποσοστό των 10% μη-κατευθυνόμενων ακμών. Βλέπουμε σε αυτό το παράδειγμα ότι με ποσοστό 10% μη-κατευθυνόμενων ακμών και 40 ενδιάμεσες ακμές μεταξύ των υπογράφων ότι ο αριθμός των μοντέλων που εντοπίστηκαν είναι πολύ κοντά στο 100 και παραμένει 100 και στη συνέχεια που το ποσοστό των μη-κατευθυνόμενων ακμών γίνεται 15%. Σε ποσοστό 10% μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων στα άλλα 4 παραδείγματα με διαφορετικούς αριθμούς ενδιάμεσων ακμών είναι πιο κάτω από 90 και περισσότερα από 70, αλλά στην συνέχεια που το ποσοστό των μη-κατευθυνόμενων ακμών γίνεται 15% και σε αυτά τα παραδείγματα ο αριθμός των μοντέλων που εντοπίστηκαν είναι 100.



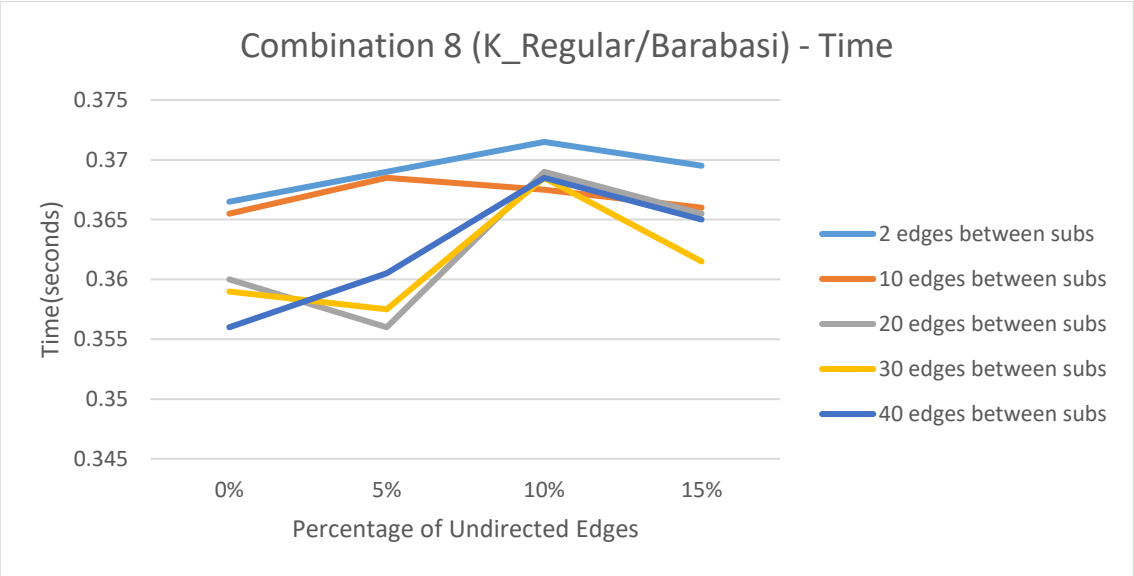
Σχήμα 5.13 Μοντέλα 7ου συνδυασμού

Στο σχήμα 5.13 βλέπουμε το μέσο χρόνο που χρειάστηκε το πρόγραμμα για να εντοπίσει τα μοντέλα. Στην αρχή για ποσοστό 0% βλέπουμε ότι σχεδόν σε όλα τα παραδείγματα ο χρόνος είναι μικρός μπορούμε όμως να δούμε ότι τα παραδείγματα με 40 ενδιάμεσες ακμές χρειαζόταν περισσότερο χρόνο από τα υπόλοιπα. Στη συνέχεια σε ποσοστό 5% μη-κατευθυνόμενων ακμών τα παραδείγματα με 2 ενδιάμεσες ακμές μεταξύ των υπογράφων βλέπουμε ότι δεν αυξήθηκε σχεδόν καθόλου ο χρόνος εντοπισμού των μοντέλων. Βλέπουμε μία μικρή αύξηση στα παραδείγματα με 10 ενδιάμεσες ακμές περίπου 50 δευτερόλεπτα. Στα παραδείγματα με 20 και 30 ενδιάμεσες ακμές παρατηρείται μία αύξηση περίπου 100 δευτερόλεπτα. Όμως στα παραδείγματα με 40 ενδιάμεσες ακμές παρατηρείται μία αύξηση περίπου 200 δευτερόλεπτα. Στη συνέχεια τα παραδείγματα με 2, 10 και 20 ενδιάμεσες ακμές βλέπουμε ότι χρειάζονται περισσότερο χρόνο από πριν αλλά υπάρχει μία πιο μικρή αύξηση. Στα παραδείγματα με 30 και 40 ενδιάμεσες ακμές παρατηρείται μία πολύ μεγάλη αύξηση που φτάνει περίπου τα 500 δευτερόλεπτα για τον εντοπισμό των μοντέλων. Τέλος, βλέπουμε μία μείωση στον χρόνο εντοπισμού των μοντέλων σε όλα τα παραδείγματα και όλα χρειάζονται περίπου 90 δευτερόλεπτα.



Σχήμα 5.14 Μοντέλο 8ου συνδυασμού

Στο σχήμα 5.14 βλέπουμε των αριθμό των μοντέλων που εντοπίστηκαν στο 8ο συνδυασμό. Ο 8ος συνδυασμός είναι συνδυασμός που έχει σαν υπογράφο του το είδος Barabasi όπως και στον 1ο και στον 4ο συνδυασμό και αυτό μας βοηθά να καταλάβουμε ότι ο αριθμός των μοντέλων σε αυτό το είδος τους γράφους αυξάνεται από το ποσοστό 5% μη-κατευθυνόμενων ακμών. Όπως και στον 1ο και στον 4ο συνδυασμό βλέπουμε ότι ο αριθμός των μοντέλων που εντοπίστηκαν σε ποσοστό 0% μη-κατευθυνόμενων ακμών είναι κοντά στο 0, για την ακρίβεια και εδώ είναι 1. Ωστόσο η αύξηση που περιμέναμε και γίνεται σε όλα τα προηγούμενα παραδείγματα ξεκινά από το ποσοστό 5% μη-κατευθυνόμενων ακμών όπως ακριβώς και στον 1ο και 4ο συνδυασμό. Στη συνέχεια σε ποσοστό 10% μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων είναι περίπου 100 για όλους τους αριθμούς ενδιάμεσων ακμών, δεν είναι ακριβώς 100 για όλα όπως παρατηρείται στην γραφική παράσταση αλλά είναι αρκετά κοντά στο 100. Τέλος, για ποσοστό 15% μη-κατευθυνόμενων ακμών ο αριθμός των μοντέλων είναι 100 για κάθε παράδειγμα που μελετήθηκε.



Σχήμα 5.15 Χρόνος 8ου συνδυασμού

Στο σχήμα 5.15 βλέπουμε τον χρόνο που χρειάστηκε το πρόγραμμα για να εντοπίσει τα μοντέλα για τον 8ο συνδυασμό. Παρατηρείται ότι το πιο αργό είναι το παράδειγμα με τις 2 ενδιάμεσες ακμές αλλά ο χρόνος που χρειάζεται είναι σχεδόν 0.37 δευτερόλεπτα που σε σύγκριση με τα προηγούμενα παραδείγματα εχτός των 1ο και τον 4ο συνδυασμό είναι πολύ μικρός. Όπως στο 1ο και στον 4ο συνδυασμό έτσι και εδώ ο χρόνος που χρειάζεται το πρόγραμμα για να εντοπίσει όλα τα μοντέλα είναι περίπου 0.37 δευτερόλεπτα που είναι ο πιο γρήγορος χρόνος που εντοπίστηκε σε όλα τα παραδείγματα που έτρεξαν.

Κεφάλαιο 6

Συμπεράσματα και Μελλοντικές Επεκτάσεις

6.1 Συμπεράσματα

Με βάση τα πειράματα που έκανα στο προηγούμενο κεφάλαιο έχω εξάγει κάποια συμπεράσματα. Υπάρχουν είδη τυχαίων γραφημάτων που δημιουργούνται με παρόμοιο τρόπο και βλέπουμε ότι έχουν την ίδια συμπεριφορά βάση των γραφικών παραστάσεων που αναλύθηκαν. Βλέπουμε ότι ο Erdos Renyi γράφος έχει κοινά χαρακτηριστικά με το Static Power Law γράφο. Παρατηρούμε ότι έχουν την ίδια συμπεριφορά στον αριθμό των μοντέλων που εντοπίζονται στους συνδυασμούς που δημιουργούνται και έχουν ως υπογράφο αυτά τα 2 είδη που ανέφερα. Παρατηρείται επίσης ίδια συμπεριφορά σε σχέση με τον χρόνο αυτών τον δύο ειδών γραφημάτων. Από την άλλη ο Barabasi γράφος βλέπουμε ότι για όλα τα παραδείγματα χρειαζόταν περίπου 0.4 δευτερόλεπτα για να εντοπιστούν τα μοντέλα ασχέτως από το ποσοστό μη-κατευθυνόμενων ακμών και από τις ενδιάμεσες ακμές.

Σε όλα τα παραδείγματα τυχαίων γραφημάτων που μελετήθηκαν βλέπουμε ότι για ποσοστό 0% μη-κατευθυνόμενων ακμών βλέπουμε ότι δεν υπάρχουν ευσταθή μοντέλα και όσο αυξάνεται το ποσοστό μη-κατευθυνόμενων ακμών αυξάνεται και ο αριθμός των μοντέλων που εντοπίζονται.

Επιπλέον στα πειράματα χρησιμοποιήθηκαν διαφορετικοί αριθμοί ενδιάμεσων ακμών μεταξύ των υπογράφων. Παρ’ όλα αυτά δεν παρατηρήθηκε κάποια διαφορά στον αριθμό των μοντέλων που εντοπίστηκαν ή στον χρόνο που χρειάστηκε το πρόγραμμα για να τα εντοπίσει. Μόνο μερικές εξαιρέσεις υπάρχουν για αυτό που αυτό μπορούμε να το αποδώσουμε στην τυχαιότητα που υπήρχε στην δημιουργία των τυχαίων γραφημάτων.

6.2 Μελλοντικές Επεκτάσεις

Στην ενότητα αυτή θα αναφέρω κάποιες επεκτάσεις που θα μπορούν να γίνουν στο μέλλον για την εκτενέστερη ανάλυση κάποιων θεωριών. Το πρόγραμμα που υλοποίησα μπορεί να δημιουργήσει και άλλα είδη γράφων από αυτά που χρησιμοποίησα στα πειράματα που ανέφερα.

Στο μέλλον θα ήταν καλό να μελετηθούν και άλλοι συνδυασμοί γραφημάτων όπως

1. Erdos-Renyi/K_Regular
2. Barabasi/K_Regular
3. Static Power Law/Barabasi
4. Forest Fire/Growing Random
5. Growing Random/Forest Fire
6. Recent Degree/Static Fitness
7. Static Fitness/Recent/Degree

και άλλα.

Επιπλέον μπορούν να μελετηθούν γράφοι με άλλα ποσοστά πυκνότητας. Στα πειράματα οι γράφοι που μελετήθηκαν είχαν πυκνότητα 5%. Θα μπορούν να μελετηθούν ακόμη συνδυασμοί γραφημάτων με πυκνότητες 15% και 25% για να μελετηθεί ο αριθμός των ευσταθών μοντέλων που μπορούν να εντοπιστούν μέσα σε ένα τέτοιο γράφημα και ο χρόνος που χρειάστηκε για να εντοπιστούν τα μοντέλα αυτά. Επίσης θα μπορούσε να χρησιμοποιηθούν διαφορετικές κορυφές για τον υπεργράφο και τους υπογράφους. Για παράδειγμα θα μπορούσε ο υπεργράφος να έχει 10 κορυφές και οι υπογράφοι 100 ή ακόμη και ο υπεργράφος να είχε 20 κορυφές και οι υπογράφοι 50.

Κεφάλαιο 7

Βιβλιογραφία

[1] Pietro Baroni, Massimiliano Giacomin, "Semantics of Abstract Argument Systems", Argumentation in Artificial Intelligence, G. Simari, I. Rahwan editors, Springer, 2009

[2] Douglas Walton, “Argumentation Theory: A Very Short Introduction” , Springer, Boston, MA, 2009

[3] Gerhard Brewka, Thomas Either and Miroslaw Truszczynski, “Answer Set Programming at a Glance”, Communications of the ACM, 2011

[4] Pietro Baroni, Martin Caminada and Massimiliano Giacomin, “An introduction to argumentation semantics”, Cambridge University Press, 2011

[5] Wikipedia, “Barabási–Albert model”
https://en.wikipedia.org/wiki/Barabási–Albert_model
Last updated on 4 January 2021

[6] Wikipedia, “Erdos-Renyi model”,
https://en.wikipedia.org/wiki/Erdős–Rényi_model
Last updated on 2 April 2021

[7] Wikipedia “Regular graph”
https://en.wikipedia.org/wiki/Regular_graph
Last update 27 March 2021

[8] Wikipedia “Forest-fire model”
https://en.wikipedia.org/wiki/Forest-fire_model
Last update on 23 September 2019

[9] Pavel L. Krapivsky and Sidney Redner, “Organization of Growing Random Networks”, 2001

[10] python-igraph API reference
https://igraph.org/python/doc/api/igraph._igraph.GraphBase.html

Παράρτημα Α

Σε αυτό το παράρτημα φαίνεται το πρόγραμμα graphgenerator.py το οποίο δημιουργεί όλα τα τυχαία γραφήματα.

```
from igraph import *
from random import *
import sys


filename = sys.argv[1]
file = open(filename, "r")
lines = file.readlines()
for line in lines:
    data = line.split(":")
    if data[0] == "hypergraph":
        hypergraph = int(data[1])
    elif data[0] == "hypergraphfile":
        hyperfile = str(data[1])
        hyperfile = hyperfile.split("\n")[0]
    elif data[0] == "edgesbetweensub":
        edgesbet = int(data[1])
    elif data[0] == "subgraph":
        subgraph = int(data[1])
    elif data[0] == "subgraphfile":
        subgraphfile = str(data[1])
        subgraphfile = subgraphfile.split("\n")[0]
    elif data[0] == "hubs":
        hubs = data[1].split("\n")[0]

    elif data[0] == "percentage":
        unper = int(data[1])



final = None
file = hyperfile
if hypergraph == 1:
    print("HyperGraph is Barabasi Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        if data[0] == "edges":

            edges = data[1].split('\n')[0]
            rangeOfEdges = edges.split('-')
            munOfEdges = randint(int(rangeOfEdges[0]),
int(rangeOfEdges[1]))
    final = Graph.Barabasi(vertices, munOfEdges, directed=True)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 2:
    print("HyperGraph is Erdos-Renyi Graph.")
    file_reader = open(file, "r")
```

```
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        if data[0] == "probability":
            p = float(data[1])
    final = Graph.Erdos_Renyi(vertices, p, directed=True)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 3:
    print("HyperGraph is K_Regular Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        if data[0] == "degree":
            degree = int(data[1])
    final = Graph.K_Regular(vertices, degree, directed=True)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 4:
    print("HyperGraph is Static Power Law Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "edges":
            edges = int(data[1])
        elif data[0] == "ex_out":
            ex_out = int(data[1])
        elif data[0] == "ex_in":
            ex_in = int(data[1])
    final = Graph.Static_Power_Law(vertices, edges, ex_out, ex_in)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 5:
    print("HyperGraph is Forest_Fire Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "fw_prob":
            fw_prob = float(data[1])
        elif data[0] == "bw_factor":
            bw_factor = int(data[1])
        elif data[0] == "ambs":
            ambs = int(data[1])
    final = Graph.Forest_Fire(vertices, fw_prob, bw_factor, ambs,
directed=True)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 6:
    print("HyperGraph is Growing Random Graph.")
```

```
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "edges":
            edges = int(data[1])
    final = Graph.Growing_Random(vertices, edges, directed=True)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 7:
    print("HyperGraph is Recent Degree Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "edges":
            edges = int(data[1])
        elif data[0] == "window":
            window = int(data[1])
    final = Graph.Recent_Degree(vertices, edges, window,
directed=True)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 8:
    print("HyperGraph is Static Fitness graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "edges":
            edges = int(data[1])
        elif data[0] == "fit_out":
            fit_out = data[1]
            fit_out = fit_out.split(" ")
            for i in range(len(fit_out)):
                fit_out[i] = int(fit_out[i])
        elif data[0] == "fit_in":
            fit_in = data[1]
            fit_in = fit_in.split(" ")
            for i in range(len(fit_in)):
                fit_in[i] = int(fit_in[i])
    final = Graph.Static_Fitness(edges, fit_out, fit_in)
    for v in final.vs:
        v["name"] = v.index
elif hypergraph == 9:
    print("HyperGraph is Degree Sequence Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "method":
            method = data[1]
        elif data[0] == "out_degree":
            out_degree = data[1]
            out_degree = out_degree.split(" ")
            for i in range(len(out_degree)):
```

```
                out_degree[i] = int(out_degree[i])
        elif data[0] == "in_degree":
            in_degree = data[1]
            in_degree = in_degree.split(" ")
            for i in range(len(in_degree)):
                in_degree[i] = int(in_degree[i])
    final = Graph.Degree_Sequence(out_degree, in_degree, method)
    for v in final.vs:
        v["name"] = v.index

file = open("theoryhyper.lp", "w")
for v in final.vs:
    arg = "arg("+str(v.index)+").\n"
    file.write(arg)

for e in final.es:
    att = "att("+str(e.source)+","+str(e.target)+").\n"
    file.write(att)

count = len(final.es)

hypervertices = len(final.vs)
file = subgraphfile

if subgraph == 1:
    print("Subgraph is Barabasi Graph.")

    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        if data[0] == "edges":
            edges = data[1].split('\n')[0]
            rangeOfEdges = edges.split('-')

    subgraphs = []
    munOfEdges = randint(int(rangeOfEdges[0]), int(rangeOfEdges[1]))
    sub = Graph.Barabasi(vertices, munOfEdges, directed=True)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        munOfEdges = randint(int(rangeOfEdges[0]),
int(rangeOfEdges[1]))
        subgraphs.append(Graph.Barabasi(vertices, munOfEdges,
directed=True))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)

        for i in range(edgessubs):
```

```
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)
elif subgraph == 2:
    print("SubGraph is Erdos-Renyi Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        if data[0] == "probability":
            p = float(data[1])
    subgraphs = []
    sub = Graph.Erdos_Renyi(vertices, p, directed=True)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.Erdos_Renyi(vertices, p,
directed=True))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)

elif subgraph == 3:
    print("SubGraph is K_Regular Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        if data[0] == "degree":
            degree = int(data[1])
```

```
    subgraphs = []
    sub = Graph.K_Regular(vertices, degree, directed=True)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.K_Regular(vertices, degree,
directed=True))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)

elif subgraph == 4:
    print("SubGraph is Static Power Law Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "edges":
            edges = int(data[1])
        elif data[0] == "ex_out":
            ex_out = int(data[1])
        elif data[0] == "ex_in":
            ex_in = int(data[1])
    subgraphs = []
    sub = Graph.Static_Power_Law(vertices, edges, ex_out, ex_in)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.Static_Power_Law(vertices, edges,
ex_out, ex_in))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)
```

```
    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)


elif subgraph == 5:
    print("SubGraph is Forest_Fire Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "fw_prob":
            fw_prob = float(data[1])
        elif data[0] == "bw_factor":
            bw_factor = int(data[1])
        elif data[0] == "ambs":
            ambs = int(data[1])
    subgraphs = []
    sub = Graph.Forest_Fire(vertices, fw_prob, bw_factor, ambs,
directed=True)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.Forest_Fire(vertices, fw_prob,
bw_factor, ambs, directed=True))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)
```

```
elif subgraph == 6:
    print("SubGraph is Growing Random Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "edges":
            edges = int(data[1])

    subgraphs = []
    sub = Graph.Growing_Random(vertices, edges, directed=True)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.Growing_Random(vertices, edges,
directed=True))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)


elif subgraph == 7:
    print("SubGraph is Recent Degree Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "vertices":
            vertices = int(data[1])
        elif data[0] == "edges":
            edges = int(data[1])
        elif data[0] == "window":
            window = int(data[1])
    subgraphs = []
    sub = Graph.Recent_Degree(vertices, edges, window, directed=True)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
```

```
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.Recent_Degree(vertices, edges, window,
directed=True))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)

elif subgraph == 8:
    print("SubGraph is Static Fitness graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "edges":
            edges = int(data[1])
        elif data[0] == "fit_out":
            fit_out = data[1]
            fit_out = fit_out.split(" ")
            for i in range(len(fit_out)):
                fit_out[i] = int(fit_out[i])
        elif data[0] == "fit_in":
            fit_in = data[1]
            fit_in = fit_in.split(" ")
            for i in range(len(fit_in)):
                fit_in[i] = int(fit_in[i])
    subgraphs = []
    sub = Graph.Static_Fitness(edges, fit_out, fit_in)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.Static_Fitness(edges, fit_out, fit_in))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
```

```
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)


elif subgraph == 9:
    print("SubGraph is Degree Sequence Graph.")
    file_reader = open(file, "r")
    lines = file_reader.readlines()
    for line in lines:
        data = line.split(":")
        if data[0] == "method":
            method = data[1]
        elif data[0] == "out_degree":
            out_degree = data[1]
            out_degree = out_degree.split(" ")
            for i in range(len(out_degree)):
                out_degree[i] = int(out_degree[i])
        elif data[0] == "in_degree":
            in_degree = data[1]
            in_degree = in_degree.split(" ")
            for i in range(len(in_degree)):
                in_degree[i] = int(in_degree[i])
    subgraphs = []
    sub = Graph.Degree_Sequence(out_degree, in_degree, method)
    num = 0
    for v in sub.vs:
        v["name"] = num
        num += 1
    for i in range(hypervertices - 1):
        subgraphs.append(Graph.Degree_Sequence(out_degree, in_degree,
method))
        for v in subgraphs[i].vs:
            v["name"] = num
            num += 1
    for g in subgraphs:
        sub = sub.union(g)
    subvertices = int(len(sub.vs) / hypervertices)

    for e in final.es:
        edgessubs = randint(1, edgesbet)
        for i in range(edgessubs):
            source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
            target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            while sub.are_connected(source, target):
                source = randint(int(e.source) * subvertices,
int(e.source) * subvertices + subvertices - 1)
                target = randint(int(e.target) * subvertices,
int(e.target) * subvertices + subvertices - 1)
            sub.add_edge(source, target)
```

```
final = sub

summary(final)

count = len(final.es)

undirectedlist = []
countundir = 0
for e in final.es:
    if final.are_connected(e.target, e.source):
        countundir += 1
        if not str(e.target)+","+str(e.source) in undirectedlist:
            undirectedlist.append(str(e.source)+","+str(e.target))

countundir = int(countundir/2)
undirected = round(unper*count/100)


directedlist = []
for e in final.es:
    if not final.are_connected(e.target,e.source):
        directedlist.append(str(e.source)+","+str(e.target))

if undirected > countundir:
    print("We have to add " + str(undirected - countundir) + "
edges.")
    added_edges = undirected - countundir
    for i in range(added_edges):
        isIllegal = True
        if len(directedlist) == 0:
            print("We only add", i, "edges")
            break
        while isIllegal:
            if len(directedlist) == 0:
                break
            num = randint(0, len(directedlist) - 1)
            edge = directedlist[num].split(",")
            source = int(edge[1])
            target = int(edge[0])
            subidsource = source // subvertices
            subidtarget = target // subvertices
            if not subidsource == subidtarget:
                edgesInDifSubs = []
                for e in final.es:
                    if e.source // subvertices == subidsource and
e.target // subvertices == subidtarget:
                        edgesInDifSubs.append(e)
                if len(edgesInDifSubs) == 0:
                    final.add_edges([(source, target)])
                    isIllegal = False
                else:

                    for ed in edgesInDifSubs:
                        if final.are_connected(ed.target, ed.source):
                            edgesInDifSubs.remove(ed)
                    if not len(edgesInDifSubs) == 0:
                        numEdge = randint(0, len(edgesInDifSubs) - 1)
                        ed = edgesInDifSubs[numEdge]
                        final.delete_edges((ed.source, ed.target))
                        final.add_edges([(source, target)])
```

```
                        isIllegal = False
                    else:
                        directedlist.remove(directedlist[num])
            else:
                edgesInSameSub = []
                for e in final.es.select(_source_eq=source):
                    edgesInSameSub.append(e)
                if len(edgesInSameSub) == 0:
                    final.add_edges([(source, target)])
                    isIllegal = False
                else:
                    for ed in edgesInSameSub:
                        if final.are_connected(ed.target, ed.source):
                            edgesInSameSub.remove(ed)
                    if not len(edgesInSameSub) == 0:
                        numEdge = randint(0, len(edgesInSameSub) - 1)
                        ed = edgesInSameSub[numEdge]
                        final.delete_edges((ed.source, ed.target))
                        final.add_edges([(source, target)])
                        isIllegal = False
                    else:
                        directedlist.remove(directedlist[num])


elif undirected < countundir:

    print("We have to delete " + str(countundir - undirected) + "
edges.")

    deleted_edges = countundir - undirected
    for i in range(deleted_edges):
        isDeleted = True
        if len(undirectedlist) == 0:
            print("We only delete", i, " edges")
            break
        while isDeleted:
            if len(undirectedlist) == 0:
                break
            num = randint(0, len(undirectedlist) - 1)
            edge = undirectedlist[num].split(",")
            sourceid = int(edge[0]) // subvertices
            targetid = int(edge[1]) // subvertices
            if sourceid == targetid:
                choice = randint(0, 1)
                if choice == 0:
                    source = int(edge[0])
                    target = int(edge[1])
                    for e in final.es.select(_target=source):
                        if e.source // subvertices == sourceid and not
final.are_connected(
                                e.target, e.source):
                            final.add_edges([(source, e.source)])
                            final.delete_edges((source, target))
                            undirectedlist.remove(undirectedlist[num])
                            isDeleted = False
                            break
                else:
                    source = int(edge[1])
                    target = int(edge[0])
                    for e in final.es.select(_target=source):
```

```
                        if e.source // subvertices == sourceid and not
final.are_connected(
                                e.target, e.source):
                            final.add_edges([(source, e.source)])
                            final.delete_edges((source, target))
                            undirectedlist.remove(undirectedlist[num])
                            isDeleted = False
                            break
                if isDeleted:
                    undirectedlist.remove(undirectedlist[num])
            else:
                choice = randint(0, 1)
                if choice == 0:
                    source = int(edge[0])
                    target = int(edge[1])
                    candidatesource = randint(sourceid * subvertices,
sourceid * subvertices + subvertices - 1)
                    candidatetarget = randint(targetid * subvertices,
targetid * subvertices + subvertices - 1)

                    while final.are_connected(candidatesource,
candidatetarget) or final.are_connected(
                            candidatetarget, candidatesource):
                        candidatesource = randint(sourceid *
subvertices, sourceid * subvertices + subvertices - 1)
                        candidatetarget = randint(targetid *
subvertices, targetid * subvertices + subvertices - 1)
                    final.add_edges([(candidatesource,
candidatetarget)])
                    final.delete_edges((source, target))
                    undirectedlist.remove(undirectedlist[num])
                    isDeleted = False
                else:
                    source = int(edge[1])
                    target = int(edge[0])
                    candidatesource = randint(sourceid * subvertices,
sourceid * subvertices + subvertices - 1)
                    candidatetarget = randint(targetid * subvertices,
targetid * subvertices + subvertices - 1)

                    while final.are_connected(candidatesource,
candidatetarget) or final.are_connected(
                            candidatetarget, candidatesource):
                        candidatesource = randint(sourceid *
subvertices, sourceid * subvertices + subvertices - 1)
                        candidatetarget = randint(targetid *
subvertices, targetid * subvertices + subvertices - 1)
                    final.add_edges([(candidatesource,
candidatetarget)])
                    final.delete_edges((source, target))
                    undirectedlist.remove(undirectedlist[num])
                    isDeleted = False
                if isDeleted:
                    undirectedlist.remove(undirectedlist[num])

else:
    print("We don't have to add/delete edges.")

print("Density: ", final.density())
file = open(sys.argv[2], "w")
```

```
for v in final.vs:
    arg = "arg("+str(v.index)+").\n"
    file.write(arg)

for e in final.es:
    att = "att("+str(e.source)+","+str(e.target)+").\n"
    file.write(att)

print(str(sys.argv[2])+" is created.")
```

Παράρτημα Β

Σε αυτό το παράρτημα φαίνεται το πρόγραμμα το οποίο χρησιμοποιεί το πρόγραμμα graphgenerator.py και δημιουργεί όσα τυχαία γραφήματα θέλει ο χρήστης

```
import sys
import os

num = 0
if len(sys.argv) == 1:
    num = 20
else:
    num = int(sys.argv[1])

com = "python graphgenerator.py input.txt "

for i in range(num):
    print("Start")
    theoryfile = "theory"+str(i) + ".lp"
    outputfile = "graph"+str(i)+".txt"
    command = com + theoryfile + " > "+outputfile
    os.system(command)
    models = 'clingo stable.lp '+theoryfile+' 500 >
model'+str(i)+'.txt'
    os.system(models)
    print("Stop")
```

Παράρτημα Γ

Σε αυτό το παράρτημα φαίνεται το πρόγραμμα συνόλου απαντήσεων το οποίο έπαιρνε σαν είσοδο την θεωρία που είχε δημιουργηθεί από το graphgenerator.py και εντόπιζε τον αριθμό των ευσταθών μοντέλων μέσα σε αυτή τη θεωρία.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Encoding for stable extensions
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% an argument x defeats an argument y if x attacks y
defeat(X,Y) :- att(X,Y), not vaf.

%% Guess a set S \subseteq A
in(X) :- not out(X), arg(X).
out(X) :- not in(X), arg(X).

%% S has to be conflict-free
:- in(X), in(Y), defeat(X,Y).

%% The argument x is defeated by the set S
defeated(X) :- in(Y), defeat(Y,X).

%% S defeats all arguments which do not belong to S
:- out(X), not defeated(X).


#show in/1.
```