ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ

ΑΥΤΟΜΑΤΗ ΚΑΤΑΤΜΗΣΗ ΠΡΟΓΡΑΜΜΑΤΩΝ SQL

Πτυχιακή εργασία της

Γρηγοριάδου Γεωργία (3254)

Επιβλέπων: Δρ. Ν. Πεταλίδης, Επιστημονικός Συνεργάτης

ΣΕΡΡΕΣ, ΟΚΤΩΒΡΙΟΣ 2017

Υπεύθυνη δήλωση

Υπεύθυνη Δήλωση: Βεβαιώνω ότι είμαι συγγραφέας αυτής της πτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην πτυχιακή εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η πτυχιακή εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Πληροφορικής ΤΕ του Τ.Ε.Ι. Κεντρικής Μακεδονίας.

Σύνοψη

Τα συστήματα λογισμικού είναι συνήθως μεγάλα και πολύπλοκα, κατά συνέπεια η κατανόηση της δομής τους είναι δύσκολη. Ένας από τους λόγους που συντελούν σε αυτή την πολυπλοκότητα είναι ότι ο πηγαίος κώδικας αποτελείται από πολλές διαφορετικού τύπου οντότητες που εξαρτώνται μεταξύ τους με περίπλοκους τρόπους. Επιπλέον από τη στιγμή που ένας μηχανικός λογισμικού κατανοήσει τη δομή ενός συστήματος είναι δύσκολο να διατηρηθεί αυτή η κατανόηση εφόσον η δομή του συστήματος τείνει να αλλάζει κατά τη συντήρηση.

Ως εκ τούτου απαιτούνται κατάλληλες γενικεύσεις για τη δομή του συστήματος. Τα εργαλεία συσταδοποίησης λογισμικού χρησιμοποιούνται για τη δημιουργία αυτών των γενικεύσεων. Στην εργασία αυτή περιγράφονται αλγόριθμοι αναζήτησης που στοχεύουν στη συσταδοποίηση του λογισμικού. Το εργαλείο που αναπτύχθηκε παράγει ένα υποσύστημα αποσύνθεσης μέσω της κατάτμησης ενός γράφου, ο οποίος αποτελείται από τις οντότητες του πηγαίου κώδικα και τις μεταξύ τους σχέσεις, χρησιμοποιώντας μια συνάρτηση καταλληλότητας για την αξιολόγηση της ποιότητας των κατατμήσεων και έναν αλγόριθμο αναζήτησης για την εύρεση ικανοποιητικής λύσης.

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος διαγραμμάτων

Κεφάλαιο 1

Εισαγωγή

Το λογισμικό υποστηρίζει πολλές από τις επιχειρήσεις αυτής της χώρας, την κυβέρνηση και κοινωνικά ιδρύματα. Δεδομένου ότι οι διαδικασίες αυτών των ιδρυμάτων αλλάζουν, έτσι πρέπει και το λογισμικό που τις υποστηρίζει. Αλλαγές σε συστήματα λογισμικού που υποστηρίζουν πολύπλοκες διαδικασίες μπορεί να είναι αρκετά δύσκολες, καθώς αυτά τα συστήματα μπορεί να είναι μεγάλα (π.χ. χιλιάδες ή ακόμα και εκατομμύρια γραμμές κώδικα) και δυναμικά.

Χωρίς σαφή εικόνα για το σχεδιασμό ενός συστήματος, ο συντηρητής λογισμικού δε μπορεί να τροποποιήσει τον πηγαίο κώδικα, χωρίς μια λεπτομερή κατανόηση της οργάνωσης του. Δεδομένου ότι οι απαιτήσεις βαριά χρησιμοποιούμενων συστημάτων λογισμικού αλλάζουν με την πάροδο του χρόνου, είναι αναπόφευκτο ότι η υιοθέτηση μιας απείθαρχης προσέγγισης για την συντήρηση του λογισμικού έχει αρνητική επίδραση στην ποιότητα της δομής του συστήματος. Τελικά, η δομή του συστήματος είναι πιθανόν να επιδεινωθεί σε σημείο όπου η οργάνωση του πηγαίου κώδικα να είναι τόσο χαοτική ώστε να πρέπει να αναθεωρηθεί ή να εγκαταλειφθεί πλήρως. Συνεπώς η δημιουργία ενός καλού νοητικού μοντέλου της δομής ενός περίπλοκου συστήματος και η διατήρηση του παρ' όλες τις αλλαγές που προκύπτουν κατά την εξέλιξη του συστήματος, είναι ένα από τα πολλά και σοβαρά προβλήματα που αντιμετωπίζουν οι σύγχρονοι προγραμματιστές λογισμικού.

Σε μια προσπάθεια αντιμετώπισης του παραπάνω προβλήματος, η ερευνητική κοινότητα της αντίστροφης μηχανικής έχει αναπτύξει τεχνικές για να αποσυνθέτουν (partition) τη δομή των συστημάτων λογισμικού σε ουσιαστικά υποσυστήματα (clusters). Τα υποσυστήματα παρέχουν στους προγραμματιστές υψηλού επιπέδου διαρθρωτικές

πληροφορίες σχετικά με τα πολυάριθμα συστατικά του λογισμικού, τις διεπαφές τους και τις διασυνδέσεις τους. Τα υποσυστήματα γενικά αποτελούνται από μια συλλογή συνεργαζόμενων πόρων του πηγαίου κώδικα που υλοποιούν ένα χαρακτηριστικό ή παρέχουν μία υπηρεσία στο υπόλοιπο σύστημα. Τα υποσυστήματα διευκολύνουν την κατανόηση του προγράμματος αντιμετωπίζοντας τα σύνολα πόρων του πηγαίου κώδικα ως οντότητες υψηλού επιπέδου.

Ο πρωταρχικός στόχος των εργαλείων συσταδοποίησης είναι να προτείνουν υποσυστήματα που παρουσιάζουν γενικεύσεις της δομής του λογισμικού. Ωστόσο, τα διάφορα εργαλεία συσταδοποίησης χρησιμοποιούν διαφορετικούς αλγορίθμους και διαφορετικά κριτήρια για το προσδιορισμό των υποσυστημάτων. Οι περισσότερες τεχνικές χρησιμοποιούν την ομοιότητα μεταξύ των στοιχείων στο πηγαίο κώδικα, βελτιστοποίηση, ή πληροφορίες σχετικά με την υλοποίηση του συστήματος, όπως ενότητες, ονόματα καταλόγων ή / και πακέτων. Σε αντίθεση με τις παραπάνω τεχνικές, η δική μας προσέγγιση αξιολογεί τη ποιότητα μιας διχοτόμησης του γραφήματος, που αντιπροσωπεύει τη δομή του λογισμικού, βάσει της σύνδεσης των επιμέρους στοιχείων του και χρησιμοποιεί ευρετικές μεθόδους ώστε να περιηγηθεί μέσα στο χώρο αναζήτησης όλων των πιθανών διχοτομήσεων του γραφήματος.

1.1 Έρευνα

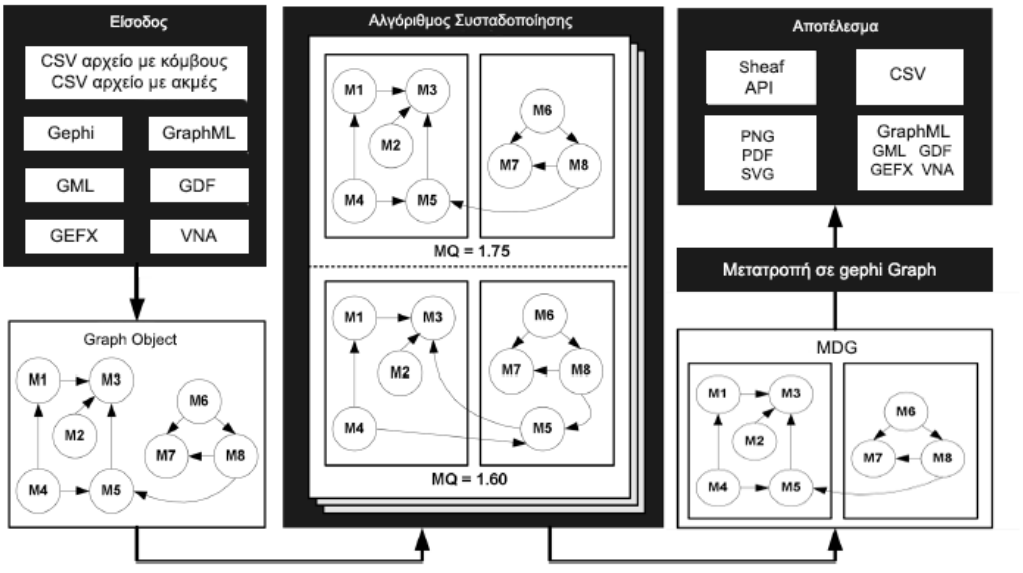
Οι αλγόριθμοι και τα εργαλεία συσταδοποίησης λογισμικού είναι χρήσιμα για την απλούστευση της συντήρησης προγραμμάτων και στη βελτίωση της κατανόησης της δομής τους. Η έρευνά μας απευθύνεται σε αυτούς τους στόχους και βασίζεται στους αλγόριθμους που περιγράφονται στο (Mitchell και Mancoridis 2006), δηλαδή τον αλγόριθμο αναρρίχησης λόφων (Hill-Climbing Algorithm) και έναν γενετικό αλγόριθμο (genetic algorithm). Δημιουργήσαμε λοιπόν ένα εργαλείο CASE (Computer Assisted Software Engineering) συσταδοποίησης PL/SQL προγραμμάτων αντίστοιχο του Bunch (Mancoridis, Mitchell, Chen κ.ά. 1999), το οποίο απευθύνεται σε προγράμματα JAVA.

Το εργαλείο χρησιμοποιεί τους γράφους που προήλθαν από parser που αναλύει κώδικα PL/SQL, ο οποίος παρουσιάζεται στο (Μιχαηλίδης 2017). Το εργαλείο προσπαθεί να χωρίσει σε συστάδες το γράφο που παρήγαγε το συγκεκριμένο εργαλείο.

Αφού τελειώσει η διαδικασία συσταδοποίησης τροφοδοτεί τη διεπαφή χρήστη με τα αποτελέσματα και την οπτικοποίηση τους και δίνει στο χρήστη τη δυνατότητα να επεξεργασθεί και να παραμετροποιήσει την οπτικοποιημένη μορφή του γράφου.

Αξίζει να σημειωθεί ότι στο (Μιχαηλίδης 2017) υλοποιήθηκε και ο γενετικός αλγόριθμος που παρουσιάζεται στο (Doval, Mancoridis και Mitchell 1999) ο οποίος είναι διαφορετικός από αυτόν που παρουσιάζεται εδώ. Περιγράφουμε τον αλγόριθμο ομαδοποίησης που χρησιμοποιήσαμε στο κεφάλαιο 2, όπως και κάποιες τροποποιήσεις που συμπεριλάβαμε ή αφαιρέσαμε στην πρωτότυπη εκδοχή του (Mitchell και Mancoridis 2006) λόγω των διαφορών ανάμεσα σε JAVA και SQL, όπως η χρήση βιβλιοθηκών και το μέγεθος των γράφων που συνήθως παράγουν. Τέλος εξετάζουμε το σχεδιασμό του εργαλείου μας στο κεφάλαιο 3, ενώ περιγράφουμε τον τρόπο χρήσης του στο κεφάλαιο 4.

1.2 Επισκόπηση της διαδικασίας συσταδοποίησης λογισμικού



Διάγραμμα 1.1: Η διαδικασία συσταδοποίησης λογισμικού

Στο διάγραμμα 1.1 απεικονίζεται η διαδικασία συσταδοποίησης λογισμικού που παρέχει το εργαλείο που αναπτύξαμε. Πάνω αριστερά έχουμε τους τύπους αρχείων που τροφοδοτούν το εργαλείο και τους οποίους είναι ικανό να αναγνωρίζει το εργαλείο

ως είσοδο. Σημαντικότεροι αυτών, τα δύο CVS αρχεία που μας παρέχει ο συντακτικός αναλυτής για SQL προγράμματα. Το πρώτο αρχείο περιέχει πληροφορίες για τους κόμβους του γράφου, ενώ το δεύτερο πληροφορίες για τις ακμές του. Όπως βλέπουμε και στο παράδειγμα του διαγράμματος 1.2 οι εν λόγω πληροφορίες αποτελούνται για τους μεν κόμβους από την ταυτότητα (id) και το όνομα του, για τις δε ακμές από το id του πηγαίου και τελικού κόμβου αντίστοιχα, από τον τύπο της ακμής (κατευθυνόμενη ή μη) καθώς και το βάρος της. Κάθε κόμβος έχει μια ταυτότητα (id) που είναι ένας μοναδικός ακέραιος αριθμός. Το βάρος μπορεί να είναι και δεκαδικός αριθμός. Το όνομα και ο τύπος ακμής παριστάνονται από αλφαριθμητικά.

Τα αρχεία Gephi (Bastian, Heymann, Jacomy κ.ά. 2009) περιέχουν πληροφορίες για ένα ολοκληρωμένο γράφο ως αντικείμενο καθώς και πληροφορίες για την οπτικοποίηση του τις οποίες μπορούμε να αξιοποιήσουμε επειδή η Gephi είναι η βιβλιοθήκη που χρησιμοποιούμε για την οπτικοποίηση και αποθήκευση του γράφου μας. Η υποστήριξη των υπόλοιπων αρχείων (GraphML, GML, GDF, GEFX και VNA) παρέχονται και αυτά από τη βιβλιοθήκη της Gephi και αποτελούν όλα αρχεία αποθήκευσης γράφων σε διαφορετικές μορφές. Επιλέγοντας λοιπόν να ανοίξουμε ένα αρχείο από τα παραπάνω μπορούμε να ομαδοποιήσουμε και γράφους που δε προήλθαν από τον SQL parser.

Κόμβοι

	A
1	Id,Name
2	1,M1
3	2,M2
4	3,M3
5	4,M4
6	5,M5
7	6,M6
8	7,M7
9	8,M8
10	

Ακμές

	A
1	Source,Target,Type,Weight
2	1,4,Directed,1
3	1,3,Directed,1
4	2,3,Directed,1
5	4,5,Directed,1
6	5,3,Directed,1
7	5,8,Directed,1
8	8,7,Directed,1
9	6,7,Directed,1
10	6,8,Directed,1

Διάγραμμα 1.2: Παράδειγμα CVS αρχείων εισόδου

Το πρώτο βήμα για την συσταδοποίηση του λογισμικού είναι να δημιουργήσουμε αντικείμενα με τις πληροφορίες που πήραμε ως είσοδο. Δημιουργούμε λοιπόν τους

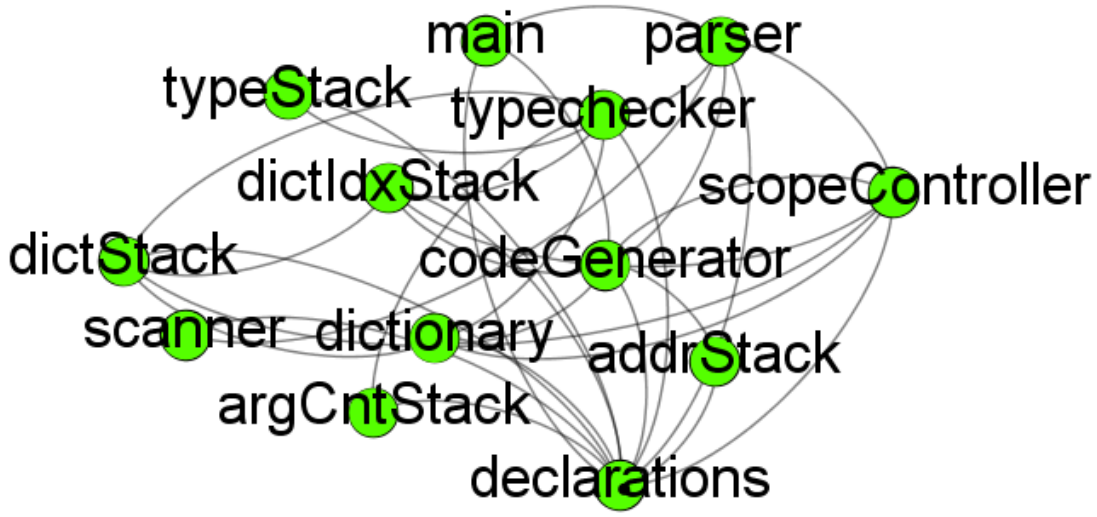
κόμβους και τις ακμές του γράφου και στη συνέχεια τον αρχικό γράφο ως την γραπτή αναπαράσταση που περιέχει το σύνολο των κόμβων και των μεταξύ τους ακμών.

Στη συνέχεια δημιουργούμε τυχαίες διαμερίσεις του γράφου ως Module Dependency Graph (MDG) αντικείμενα. Ορίζουμε τον MDG αναλυτικότερα στο κεφάλαιο 2. Προς το παρόν θεωρούμε τον MDG ως ένα γράφο που αναπαριστά τα επιμέρους συστατικά (modules) του συστήματος ως κόμβους και τις εξαρτήσεις μεταξύ τους ως σταθμισμένες κατευθυνόμενες ακμές. Η παραπάνω πληροφορία που έχει αυτό το αντικείμενο είναι ότι περιέχει επιπλέον το σύνολο των υποσυστημάτων (clusters) στους οποίους κατανέμονται οι κόμβοι και η ποιότητα κατάτμησης που αντιστοιχεί σε κάθε υποσύστημα.

Ο στόχος του εργαλείου που αναπτύξαμε είναι να αναδείξει υποσυστήματα που έχουν νόημα σύμφωνα με τις εξαρτήσεις ανάμεσα στα συστατικά του υπό εξέταση λογισμικού. Υποβάλουμε λοιπόν αρχικά τις τυχαίες διαμερίσεις (MDGs) που αναφέρθηκαν παραπάνω στον αλγόριθμο συσταδοποίησης λογισμικού. Στη συνέχεια ο αλγόριθμος θα μας παρέχει τη βέλτιστη διαμέριση του γράφου σύμφωνα με τη ποιότητα κατάτμησης.

Εφόσον έχει βρεθεί η βέλτιστη διαμέριση του γράφου, χρησιμοποιούμε κατάλληλη βιβλιοθήκη σχεδιασμού γράφων, στη δική μας περίπτωση το Gephi για την οπτικοποίηση αλλά αργότερα και για την αποθήκευση των αποτελεσμάτων. Τέλος αξίζει να σημειωθεί ότι ο χρήστης έχει τη δυνατότητα να επεξεργασθεί και να παραμετροποιήσει το οπτικό αποτέλεσμα μέσω του user interface που αναπτύξαμε και να αποθηκεύσει το αποτέλεσμα σε μορφή γράφου, εικόνας ή κειμένου.

Στο διάγραμμα 1.3 βλέπουμε ένα παράδειγμα γράφου για ένα μικρό compiler, και στο διάγραμμα 1.4 τη βέλτιστη διαμέριση του που παρήγαγε το εργαλείο που αναπτύξαμε, η οποία και ταυτίζεται με εκείνη του εργαλείου Bunch. Παρατηρούμε ότι έχει δημιουργήσει τέσσερα υποσυστήματα για την αναπαράσταση της γενικευμένης δομής του compiler. Συγκεκριμένα υπάρχουν υποσυστήματα για την παραγωγή κώδικα (code generation), διαχείριση εύρους ορισμού μεταβλητών (scope management), έλεγχο τύπων (type checking) και συντακτική ανάλυση (parsing).



Διάγραμμα 1.3: Ο γράφος για ένα μικρό compiler (Αναπαρωγή από το (Mitchell και Mancoridis 2006))

1.3 Περίγραμμα εργασίας

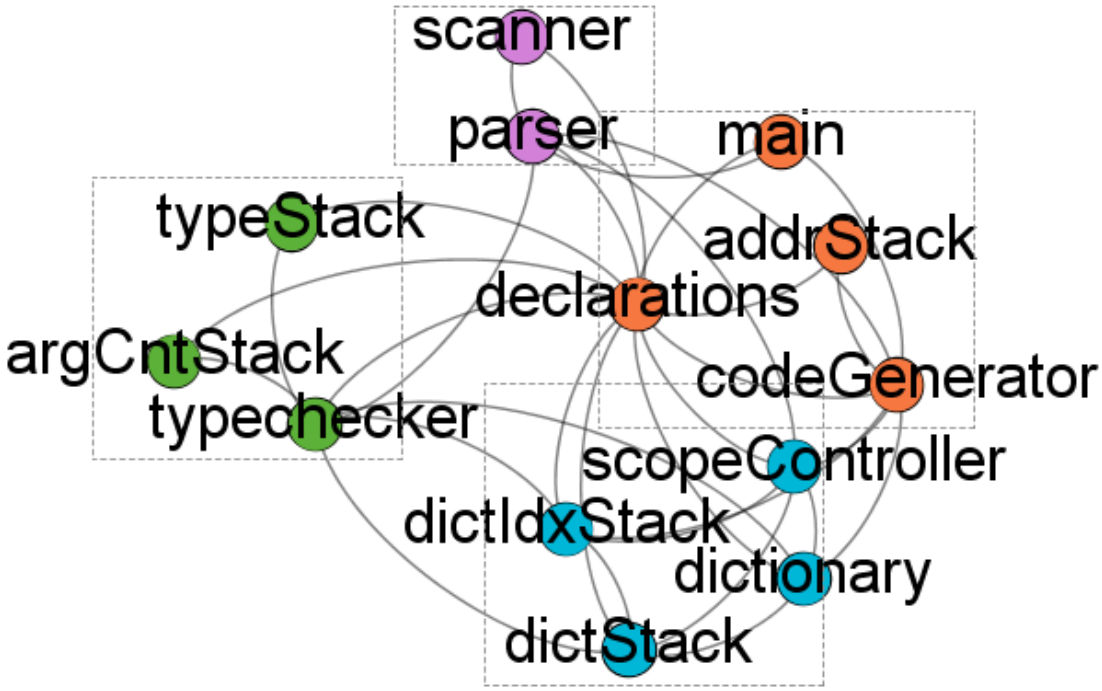
Σε αυτό το κεφάλαιο παρουσιάσαμε το πρόβλημα της συσταδοποίησης λογισμικού και περιγράψαμε την προσέγγισή μας προς την αντιμετώπιση αυτού του προβλήματος. Το υπόλοιπο αυτής της ενότητας παρέχει μια επισκόπηση των κεφαλαίων αυτής της εργασίας.

- Κεφάλαιο 2 - Αλγόριθμοι συσταδοποίησης λογισμικού

  Στο κεφάλαιο αυτό αναφέρουμε ότι αξιολογούμε τις διαμερίσεις ενός MDG βάσει της ποιότητας κατάτμησης του (MQ) και τον τρόπο υπολογισμού της. Επίσης περιγράφουμε ένα αλγόριθμο εξαντλητικής αναζήτησης και ένα ευρετικό αλγόριθμο. Αυτός ο αλγόριθμος δέχεται μια γραφική αναπαράσταση της δομής ενός συστήματος λογισμικού ως είσοδο και παράγει ένα γράφημα διαμερισμένο σε επιμέρους υποσυστήματα. Επίσης αναφέρουμε βελτιώσεις και τροποποιήσεις τις πρωτότυπης εκδοχής του αλγορίθμου.

- Κεφάλαιο 3 - Ο Σχεδιασμός του εργαλείου Sheaf

  Περιγράφεται ο σχεδιασμός του εργαλείου Sheaf και των εργαλείων που χρησιμοποιήθηκαν για την υλοποίηση του. Δίνεται έμφαση στην πτυχές σχεδιασμού του Sheaf που υποστηρίζουν την επέκταση του και την υλοποίηση επιπλέον αλγορίθμων.



Διάγραμμα 1.4: Η βέλτιστη διαμέριση MDG για ένα μικρό compiler (Αναπαρωγή από το (Mitchell και Mancoridis 2006))

- Κεφάλαιο 4 - Εγχειρίδιο χρήσης του εργαλείου Sheaf

  Σε αυτό το κεφάλαιο παρουσιάζουμε το εργαλείο Sheaf που αναπτύξαμε και ένα μικρό παράδειγμα χρήσης του. Επίσης παρέχονται οδηγίες για τα διάφορα σενάρια χρήσης για τη σωστή και εύκολη εκμετάλλευση των δυνατοτήτων του από το χρήστη.

- Κεφάλαιο 5 - Επικύρωση αποτελεσμάτων και Συμπεράσματα

  Το τελευταίο κεφάλαιο της παρούσας εργασίας συνοψίζει τη συμβολή του έργου μας στον τομέα της συσταδοποίησης λογισμικού και τα συμπεράσματα που καταλήξαμε κατά τη συγγραφή της, επίσης συμπεριλαμβάνει την επικύρωση των αποτελεσμάτων του αλγόριθμου.

Κεφάλαιο 2

Αλγόριθμοι Συσταδοποίησης Λογισμικού

Η προσέγγιση που παρουσιάζεται για τη συσταδοποίηση λογισμικού είναι ανεξάρτητη από οποιαδήποτε γλώσσα προγραμματισμού. Για το λόγο αυτό χρησιμοποιούμε εργαλεία ανάλυσης πηγαίου κώδικα ώστε να μεταφέρουμε τη δομή του συστήματος, τις συνιστώσες του και τις μεταξύ τους σχέσεις σε ένα κατευθυνόμενο γράφο. Αναφερόμαστε σε αυτό το γράφο ως Module Dependency Graph (MDG). Η προσέγγιση αυτή, για την επίλυση του προβλήματος συσταδοποίησης, αναφέρεται ανεπίσημα ως "η εξεύρεση μιας καλής διαμερίσης για ένα MDG γράφο".

Χρησιμοποιούμε τον όρο διαμέριση με την έννοια της Θεωρίας Γράφων, η οποία είναι, η αποσύνθεση ενός συνόλου στοιχείων (δηλαδή, όλοι οι κόμβοι ενός γραφήματος) σε διαχωρισμένα μεταξύ τους σύνολα (δηλαδή υποσυστήματα). Με μία καλή διαμέριση εννοούμε μία διαμέριση όπου οι εξαιρετικά αλληλοεξαρτώμενες ενότητες (κόμβοι) ομαδοποιούνται στο ίδιο υποσύστημα (cluster) και, αντιστρόφως, οι ανεξάρτητες μονάδες ανατίθενται σε ξεχωριστά υποσυστήματα.

Επισήμως, ο $MDG = (V, E)$ είναι ένα γράφημα όπου V είναι το σύνολο των ονομασμένων κόμβων ενός συστήματος λογισμικού, καί $E \subseteq V \times V$ είναι το σύνολο των διατεταγμένων ζευγών $\langle u, v \rangle$ που αντιπροσωπεύουν τις σχέσεις μεταξύ των κόμβων u και v.

Δεδομένου ενός $MDG, S = (V, E)$, μία διαμέριση του G σε n υποσυστήματα ορίζεται επίσημα ως $\Pi_S = S_1, S_2, ..., S_n$, όπου κάθε $S_i((1 \leq i \leq n) \wedge (n \leq |V|))$ είναι ένα υποσύστημα του διαμερισμένου γράφου. Ειδικά:

$$S = (V, E),\ \Pi_S = \bigcup_{i=1}^{n} S_i$$

$$S_i = (V_i, E_i)$$

$$\bigcup_{i=1}^{n} V_i = V$$

$$\forall((1 \leq i, j \leq n) \wedge (i \neq j), V_i \cap V_j = \emptyset$$

$$E_i = \{\langle u, \upsilon\rangle \in E | u \in V_i \wedge \upsilon \in V\}$$

Εάν η διαμέριση Π_S είναι το σύνολο των υποσυστημάτων του MDG, κάθε υποσύστημα S_i περιέχει ένα μη επικαλυπτόμενο σύνολο των κόμβων από το V και ακμές από το E. Ο αριθμός των υποσυστημάτων σε μία διαμέριση κυμαίνεται από το 1 (ένα ενιαίο υποσύστημα που περιέχει όλους τους κόμβους) έως $|V|$ (κάθε κόμβος του γράφου αποτελεί ένα μονομερές υποσύστημα). Μία διαμέριση του MDG με k $(1 \leq k \leq |V|)$ μη-κενά υποσυστήματα ονομάζεται k-διαμέριση του MDG.

Δεδομένου ενός MDG που περιέχει $n = |V|$ κόμβους, και k υποσυστήματα, ο αριθμός S_n, διακριτών k-διαμερίσεων του MDG ικανοποιεί την εξίσωση επανάληψης:

$$S_{n,k} = \begin{cases} 1 & k = 1 k = n \\ S_{n-1,k-1} + k S_{n-1,k} \end{cases}$$

Οι καταχωρήσεις $S_{n,k}$ ονομάζονται αριθμοί Stirling και αυξάνονται με γεωμετρική πρόοδο σε σχέση με το μέγεθος του συνόλου V. Για παράδειγμα, ένα γράφημα 5 κόμβων έχει 52 διακριτές διαμερίσεις, ενώ ένα με 15 κόμβους διαθέτει 1.382.958.545 ξεχωριστές διαμερίσεις. Το παραπάνω κείμενο αποτελεί μετάφραση του αντίστοιχου στο Mitchell και Mancoridis 2002a, εφόσον δεν υπήρχαν να συμπληρωθούν περαιτέρω στοιχεία από συναφή άρθρα και δημοσιεύσεις.

2.1 Αξιολόγηση διαμερίσεων του MDG

Ο πρωταρχικός στόχος των αλγορίθμων συσταδοποίησης λογισμικού είναι να προτείνει υποσυστήματα (δηλαδή, μη-επικαλυπτόμενα σύνολα στοιχείων) που εκθέτουν γενικεύσεις της δομής του λογισμικού. Η εύρεση μίας καλής διαμέρισης περιλαμβάνει

την πλοήγηση μέσα στο χώρο αναζήτησης όλων των πιθανών διαμερίσεων του MDG με συστηματικό τρόπο. Για να επιτευχθεί αυτό το έργο αποτελεσματικά, οι αλγόριθμοι συσταδοποίησης αντιμετωπίζουν το διαμερισμό του γράφου (clustering) ως ένα πρόβλημα αναζήτησης. Ο στόχος της έρευνας είναι να μεγιστοποιήσει την αξία της αντικειμενικής συνάρτησης, η οποία ονομάζεται ποιότητα κατάτμησης (Modularization Quality - MQ).

Η MQ καθορίζει την ποιότητα ενός τμήματος του MDG ποσοτικά ως το συμβιβασμό μεταξύ της εξωτερικής συνδεσιμότητας (δηλαδή, εξαρτήσεις μεταξύ των στοιχείων από δύο διακριτά υποσυστήματα) και της εσωτερικής συνδεσιμότητας (δηλαδή, τις εξαρτήσεις μεταξύ των στοιχείων του ίδιου υποσυστήματος). Αυτός ο συμβιβασμός βασίζεται στην υπόθεση ότι ένα καλά σχεδιασμένο σύστημα λογισμικού οργανώνεται σε συνεκτικά υποσυστήματα που είναι χαλαρά συνδεδεμένα μεταξύ τους. Ως εκ τούτου, η MQ έχει σχεδιαστεί για να ανταμείψει τη δημιουργία εξαιρετικά συνεκτικών υποσυστημάτων που δεν συνδέονται υπερβολικά. Κατά παραδοχή όσο μεγαλύτερη είναι η MQ, τόσο πιο κοντά είναι η διαμέριση του MDG στην δομή του επιθυμητού υποσυστήματος.

Ένας τρόπος για να βρεθεί η καλύτερη διαμέριση ενός MDG είναι να εκτελεστεί μια εξαντλητική αναζήτηση μέσω όλων των έγκυρων διαμερίσεων, και να επιλεχθεί εκείνη με τη μεγαλύτερη τιμή της MQ. Ωστόσο, αυτό είναι συχνά αδύνατο επειδή ο αριθμός των τρόπων που μπορεί να κατανεμηθεί ο MDG αυξάνεται εκθετικά σε σχέση με τον αριθμό των κόμβων του (modules). Επειδή η ανακάλυψη της βέλτιστης διαμέρισης ενός MDG είναι εφικτή μόνο για μικρά συστήματα λογισμικού (π.χ., με λιγότερους από 15 κόμβους), στρέφουμε τη προσοχή μας στη χρήση ευρετικών αλγορίθμων αναζήτησης που είναι ικανοί στην γρήγορη εξόρυξη προσεγγιστικών αποτελεσμάτων.

2.2 Υπολογισμός της ποιότητας κατάτμησης (MQ)

Στη παραπάνω ενότητα αναφερθήκαμε στον υπολογισμό της MQ ως την αντικειμενική συνάρτηση της διαδικασίας αναζήτησης , ενώ την ορίσαμε και ως ο υπολογισμός της ποιότητας μιας συγκεκριμένης κατάτμησης του συστήματος. Η MQ υπολογίζει ποσοτικά το βαθμό της εξωτερικής συνδεσιμότητας και της εσωτερικής συνδεσιμότητας. Έχει σχεδιαστεί για να παράγει μεγαλύτερες τιμές όσο ο βαθμός της εσωτερικής

συνδεσιμότητας αυξάνεται και ο βαθμός της εξωτερικής συνδεσιμότητας μειώνεται. Παρακάτω παραθέτουμε τις έννοιες της εσωτερικής και εξωτερικής συνδεσιμότητας, όπως αυτές ορίζονται στο (Mancoridis, Mitchell, Rorres κ.ά. 1998).

2.2.1 Εσωτερική Συνδεσιμότητα

Η εσωτερική συνδεσιμότητα (Intra-Connectivity) ορίζεται ως το μέτρο της συνδεσιμότητας μεταξύ των συστατικών που είναι ομαδοποιημένα στο ίδιο υποσύστημα. Ένας υψηλός βαθμός εσωτερικής συνδεσιμότητας υποδεικνύει μία καλή κατάτμηση, επειδή τα στοιχεία που έχουν ομαδοποιηθεί σε ένα κοινό υποσύστημα μοιράζονται πολλά χαρακτηριστικά επιπέδου λογισμικού. Ένας χαμηλός βαθμός ενδόσυνδεσιμότητας υποδεικνύει κακή κατάτμηση του υποσυστήματος, διότι τα στοιχεία που έχουν ανατεθεί σε ένα κοινό υποσύστημα παρουσιάζουν χαμηλή συνοχή μεταξύ τους.

Η τιμή της εσωτερικής συνδεσιμότητας A_i του υποσυστήματος i το οποίο αποτελείται από N_i συστατικά και μ_i εσωτερικές ακμές εξάρτησης, δίνεται από τον τύπο:

$$A_i = \frac{\mu_i}{N_i^2}$$

Αυτή η μέτρηση είναι ένα κομμάτι του μέγιστου αριθμού των εσωτερικών ακμών εξάρτησης που μπορεί να υπάρχουν για υποσύστημα i, που είναι N_i^2 . Η τιμή του A_i οριοθετείται μεταξύ των τιμών 0 και 1. A_i είναι 0 όταν τα στοιχεία σε ένα υποσύστημα δεν μοιράζονται πόρους σε επίπεδο λογισμικού, A_i είναι ίσο με 1 όταν κάθε μονάδα σε ένα υποσύστημα χρησιμοποιεί ένα πόρο λογισμικού από όλες τις άλλες μονάδες στο υποσύστημα του (δηλαδή, τα στοιχεία και οι εξαρτήσεις εντός ενός υποσυστήματος αποτελούν ένα πλήρες γράφο).

2.2.2 Εξωτερική Συνδεσιμότητα

Η εξωτερική συνδεσιμότητα (Inter-Connectivity) ορίζεται ως το μέτρο της συνδεσιμότητας ανάμεσα σε δύο διακριτά υποσυστήματα. Ένας υψηλός βαθμός εξωτερικής συνδεσιμότητας αποτελεί ένδειξη κακής κατάτμησης υποσυστήματος. Ένας μεγάλος αριθμός αλληλεξαρτήσεων περιπλέκει τη διαδικασία συντήρησης λογισμικού επειδή οι αλλαγές σε μια λειτουργική μονάδα μπορεί να επηρεάσουν πολλά άλλα μέρη του συστήματος, λόγω των αλληλεπιδράσεων τους με το υποσύστημα. Ένας μικρός βαθ-

μός διασυνδεσιμότητας είναι ένα επιθυμητό γνώρισμα της δομής ενός συστήματος και είναι μια ένδειξη ότι τα μεμονωμένα υποσυστήματα του συστήματος είναι, σε μεγάλο βαθμό, ανεξάρτητα. Ως εκ τούτου, οι αλλαγές που εφαρμόζονται σε μία μονάδα είναι πιθανό να περιορίζονται μόνο στο υποσύστημα της, κάτι που μειώνει την πιθανότητα δημιουργίας σφαλμάτων σε άλλα μέρη του συστήματος όταν εφαρμόζεται μια αλλαγή.

Η διασυνδεσιμότητα E_{ij} μεταξύ των υποσυστημάτων i και j που αποτελείται από N_i και N_j στοιχεία, αντίστοιχα, με ϵ_i εξωτερικές ακμές εξάρτησης δίνεται από τον τύπο:

$$E_{i,j} = \begin{cases} 0 & , i = j \\ \frac{\epsilon_{i,j}}{2N_iN_j}, & i \neq j \end{cases}$$

Αυτή η μέτρηση της διασυνδεσιμότητας μας είναι ένα κομμάτι του μέγιστου αριθμού των ακμών εξαρτήσεως μεταξύ των υποσυστημάτων i και j ($2 \times N_i \times N_j$). Η τιμή E_{ij} οριοθετείται μεταξύ των τιμών 0 και 1. E_{ij} είναι ίση με 0 όταν δεν υπάρχουν εξαρτήσεις μεταξύ των υποσυστημάτων i και j ενώ E_{ij} είναι 1 όταν κάθε μονάδα στο υποσύστημα i εξαρτάται από όλες τις μονάδες στο υποσύστημα j και αντίστροφα.

2.2.3 Υπολογισμός της BasicMQ

Εφόσον έχουμε περιγράψει τις έννοιες της ενδοσυνδεσιμότητας και της διασυνδεσιμότητας, ο υπολογισμός της BasicMQ η οποία είναι μία διαμέριση του MDG σε k-υποσυστήματα, όπου A_i είναι η τιμή της εσωτερικής συνδεσιμότητας του υποσυστήματος i και E_{ij} η τιμή της εξωτερικής συνδεσιμότητας μεταξύ των υποσυστημάτων i και j, ορίζεται ως:

$$BasicMQ = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} A_i - \frac{1}{\frac{k(k-1)}{2}} \sum_{i,j=1}^{k} E_{i,j}, & k > 1 \\ A_i & , k = 1 \end{cases}$$

Ο υπολογισμός της BasicMQ παρουσιάζει τον συμβιβασμό ανάμεσα στην ενδοσυνδεσιμότητα και την διασυνδεσιμότητα ανταμείβοντας τη δημιουργία εξαιρετικά συνεκτικών υποσυστημάτων. Αυτός ο συμβιβασμός ορίζεται ως η αφαίρεση της μέσης τιμής της εξωτερικής συνδεσιμότητας από την μέση τιμή της εσωτερικής συνδεσιμότητας.

Χρησιμοποιούνται οι μέσες τιμές των A και E για την εξασφάλιση της συνοχής

μεταξύ των μονάδων κατά την αφαίρεση καθώς το σύνολο της ενδοσυνδεσιμότητας βασίζεται στον αριθμό των υποσυστημάτων (k), και το σύνολο της διασυνδεσιμότητας βασίζεται στον αριθμό των διακριτών ζευγών υποσυστημάτων ($\frac{k(k-1)}{2}$). Η τιμή της BasicMQ ορίζεται ανάμεσα στο -1 (καμία συνεκτικότητα μέσα στα υποσυστήματα) και 1 (καμία συνοχή ανάμεσα στα υποσυστήματα).

Ο υπολογισμός της BasicMQ έχει δύο σημαντικά μειονεκτήματα. Πρώτον, ο απαιτούμενος χρόνος για τον υπολογισμό είναι μεγάλος συνεπώς και η απόδοση του χαρακτηρίζεται ως κακή , γεγονός που περιορίζει τη χρήση του σε μικρά συστήματα.

Το δεύτερο πρόβλημα με τον υπολογισμό της BasicMQ είναι ότι ο σχεδιασμός του δεν μπορεί να υποστηρίξει MDG που έχουν ακμές με βάρη. Αυτός ο περιορισμός βασίζεται στον τρόπο που υπολογίζονται τόσο η ενδο-συνδεσιμότητα όσο και η διασυνδεσιμότητα . Συγκεκριμένα, η ενδο-συνδεσιμότητα και η διασυνδεσιμότητα είναι αναλογίες των πραγματικών ακμών επί του συνόλου των πιθανών αριθμών ακμών, και δεν βασίζονται στο πραγματικό βάρος των μεμονωμένων ακμών. Εάν έχουν καθοριστεί τα βάρη των ακμών είναι αδύνατον να καθορίσουμε τον παρονομαστή για τις συναρτήσεις της διασυνδεσιμότητας και της ενδο-συνδεσιμότητας.

2.2.4 TurboMQ

Η TurboMQ σχεδιάστηκε για να ξεπεραστούν οι δύο περιορισμοί της BasicMQ. Συγκεκριμένα, η TurboMQ υποστηρίζει MDG που έχουν ακμές με βάρη, και είναι πολύ πιο γρήγορη από την BasicMQ.

Επισήμως, ο υπολογισμός της TurboMQ για MDG το οποίο αποτελείται από k-υποσυστήματα υπολογίζεται αθροίζοντας το Cluster Factor (CF) για κάθε υποσύστημα του διαμερισμένου MDG.

Ο Cluster Factor, CF_i, για υποσύστημα i $(1 \leq i \leq k)$ ορίζεται ως μια κανονικοποιημένη αναλογία μεταξύ του συνολικού βάρους των εσωτερικών ακμών (ακμών εντός του υποσυστήματος) και το ήμισυ του συνολικού βάρους των εξωτερικών ακμών (ακμές που εισέρχονται ή εξέρχονται στο υποσύστημα). Το βάρος των εξωτερικών ακμών χωρίζεται κατά το ήμισυ, προκειμένου να εφαρμοσθεί μια ίση ποινή και για τα δύο υποσυστήματα που συνδέονται από μια εξωτερική ακμή.

Παρόμοια με τη συνάρτηση της BasicMQ αναφερόμαστε στις εσωτερικές ακμές ενός υποσυστήματος ως intra-edges (μ_i), και στις ακμές μεταξύ δύο διακριτών υποσυ-

στημάτων i και j ως inter-edges ($\epsilon_{i,j}$ και $\epsilon_{j,i}$ αντίστοιχα). Εάν τα βάρη των ακμών δεν παρέχονται από τον MDG, υποθέτουμε ότι κάθε ακμή έχει βάρος 1. Επίσης, σημειώνουμε ότι $\epsilon_{i,j} = 0$ και $\epsilon_{j,i} = 0$, όταν $i = j$. Παρακάτω ορίζεται η τιμή της TurboMQ ως:

$$TurboMQ = \sum_{i=1}^{k} CF_i$$

$$CF_i = \begin{cases} 0 & \mu_i = 0 \\ \frac{2\mu_i}{2\mu_i + \sum_{j=1, j\neq i}^{k} \epsilon_{ij} + \epsilon_{ji}} & otherwise \end{cases}$$

Οι τύποι που χρησιμοποιήθηκαν για τη περιγραφή τόσο της TurboMQ όσο και της BasicMQ πάρθηκαν από το Mitchell 2002.

2.3 Αλγόριθμοι

Προηγουμένως παρουσιάσαμε τον ορισμό του MDG, πώς μπορεί να κατασκευασθεί ο MDG από τον πηγαίο κώδικα, και πώς να μετρηθεί η «ποιότητα» μιας διαμέρισης του MDG χρησιμοποιώντας αντικειμενικές συναρτήσεις. Το υπόλοιπο αυτής της ενότητας περιγράφει τους αλγορίθμους συσταδοποίησης που αναφέρονται στο (Mitchell και Mancoridis 2002a) και εκείνον που έχουμε εμείς δημιουργήσει.

Αρχίζουμε αναφέροντας τον εξαντλητικό αλγόριθμο αναζήτησης, και στη συνέχεια παρουσιάζουμε έναν αλγόριθμο αναζήτησης αναρρίχησης λόφων και τις βελτιώσεις που εισήγαγαν ο Mitchell και Mancoridis. Τέλος ορίζουμε τον προσαρμοσμένο αλγόριθμο αναρρίχησης λόφων που δημιουργήσαμε.

2.3.1 Αλγόριθμος εξαντλητικής αναζήτησης

Ο αλγόριθμος εξαντλητικής αναζήτησης λειτουργεί υπολογίζοντας την MQ για όλες τις πιθανές διαμερίσεις του MDG. Η διαμέριση με τη μεγαλύτερη MQ επιστρέφεται. Ο αλγόριθμος 1 δείχνει τον εξαντλητικό αλγόριθμο συσταδοποίησης όπως αυτός παρουσιάζεται στο (Mitchell και Mancoridis 2002a).

Στις προηγούμενες ενότητες δείξαμε πώς μπορεί να προσδιοριστεί ο συνολικός αριθμός των διαμερίσεων του MDG. Έστω $S_{n,k}$ είναι ένας αριθμός Stirling του δεύ-

τερου είδους, ο οποίος δείχνει τον αριθμό των τρόπων με τον οποίο μπορούν να διαμεριστούν n κόμβοι σε k μη κενά υποσυστήματα. Αν ο k αντιπροσωπεύει τον αριθμό των υποσυστημάτων, γνωρίζουμε ότι $(1 \leq k \leq n)$, διότι ο συνολικός αριθμός των υποσυστημάτων σε ένα χωρισμένο MDG μπορεί να κυμαίνεται από το 1 (ένα ενιαίο σύμπλεγμα που περιέχει όλες τις ενότητες) έως n (n συστάδες που το καθένα περιέχει μια μονάδα). Είναι γνωστό ότι το άθροισμα των αριθμών Stirling του δεύτερου είδους μπορεί να οριοθετείτε χρησιμοποιώντας την ανισότητα:

$$2^{n-1} < \sum_{k=1}^{n} S_{n,k} < n!$$

Επειδή ο αλγόριθμος εξαντλητικής αναζήτησης πρέπει να εξετάσει ένα πολύ μεγάλο αριθμό διαμερίσεων, είναι κατάλληλος μόνο για πολύ μικρά συστήματα με λιγότερες από 15 μονάδες. Πέραν αυτού του αριθμού, ο χώρος αναζήτησης (αριθμός k-διαμερίσεων του MDG) γίνεται τόσο μεγάλος που δεν μπορεί να διερευνηθεί σε ένα εύλογο χρονικό διάστημα.

Αλγόριθμος 1 Αλγόριθμος Εξαντλητικής Αναζήτησης

Εξαντλητική Αναζήτηση(MDG M)
Έστω S = (M1,M2,...,Mn), το σύνολο όλων των έγκυρων διαμερίσεων το M όπου κάθε M_i είναι ένα στοιχείο του συστήματος λογισμικού.
Έστω B η διαμέριση του M με την υψηλότερη MQ, αρχικοποιημένη στο $\emptyset$
Έστω $max_{mq} = -\infty$
Για κάθε διαμέριση $(s \in S)$ **επανέλαβε**
Έστω mqs = MQ(s)
Αν (mqs > maxmq) **τότε**
B ← s
maxmq ← mqs
ΤέλοςΑν
ΤέλοςΕπανάληψης
επέστρεψε (διαμέριση B).

2.3.2 Αλγόριθμος αναρρίχησης λόφων

Ο αλγόριθμος εξαντλητικής αναζήτησης που συζητήθηκε στην προηγούμενη ενότητα προσδιορίζει το ”καλύτερο” αποτέλεσμα με βάση την τιμή της αντικειμενικής συνάρτησης της MQ. Ωστόσο, ο χώρος αναζήτησης που απαιτείται για να απαριθμήσει όλες τις πιθανές διαμερίσεις ενός συστήματος λογισμικού γίνεται απαγορευτικά μεγάλος όσο αυξάνεται ο αριθμός των στοιχείων στο σύστημα. Για την αντιμετώπιση

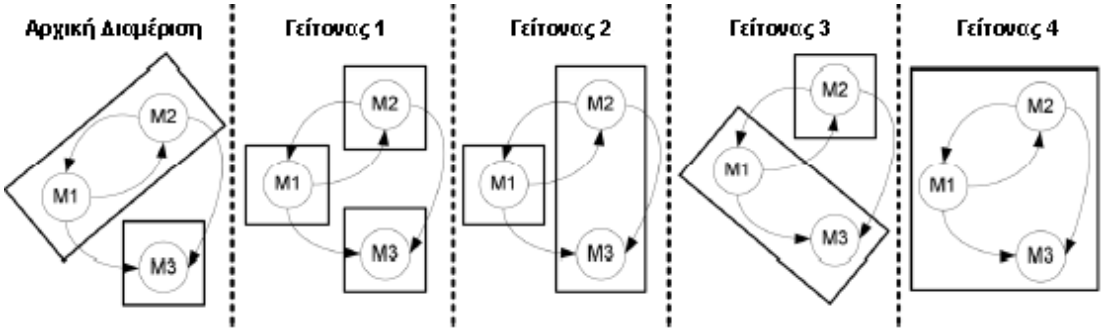
αυτού του προβλήματος, εφαρμόστηκε μια ευρετική στρατηγική αναζήτησης, με βάση τις τεχνικές αναζήτησης αναρρίχησης λόφου, που ανακαλύπτει γρήγορα ένα αποδεκτό υπο-βέλτιστο αποτέλεσμα συσταδοποίησης.

Οι αλγόριθμοι αναρρίχησης λόφων συσταδοποίησης (hill climbing clustering algorithms) ξεκινούν με μια τυχαία διαμέριση του MDG. Στη συνέχεια αναδιατάσσονται συστηματικά σε μια προσπάθεια να βρουν μία ”βελτιωμένη” διαμέριση με υψηλότερη MQ. Αν μία καλύτερη διαμέριση βρεθεί, η διαδικασία επαναλαμβάνεται, χρησιμοποιώντας τη βελτιωμένη διαμέριση ως βάση για την εξεύρεση ακόμα καλύτερων διαμερίσεων. Αυτή η προσέγγιση αναρρίχησης λόφων συγκλίνει τελικά, όταν δεν μπορούν να βρεθούν επιπλέον διαμερίσεις με υψηλότερη MQ.

Κάθε τυχαία αρχική διαμέριση του MDG συγκλίνει τελικά σε ένα τοπικό μέγιστο. Δυστυχώς, υπάρχουν αρχικές διαμερίσεις των MDG οι οποίες δεν παράγουν αποδεκτές βέλτιστες λύσεις. Αυτό το πρόβλημα, αντιμετωπίζεται με τη δημιουργία ενός αρχικού πληθυσμού (δηλαδή, μία συλλογή) τυχαίων διαμερίσεων. Οι αλγόριθμοι αναρρίχησης λόφων συγκεντρώνουν κάθε μία από τις τυχαίες διαμερίσεις του πληθυσμού, και επιλέγεται το αποτέλεσμα με τη μεγαλύτερη MQ ως υπό-βέλτιστη λύση. Καθώς το μέγεθος του πληθυσμού αυξάνει, η πιθανότητα να βρεθεί μια καλή υπό-βέλτιστη λύση, επίσης αυξάνει.

Γειτονικές διαμερίσεις (Neighbouring Partitions)

Οι αλγόριθμοι αναρρίχησης λόφων κινούν στοιχεία μεταξύ των υποσύνολων των διαμερίσεων σε μια προσπάθεια να βελτιώσουν την MQ. Αυτή η διεργασία επιτυγχάνεται με τη δημιουργία ενός συνόλου γειτονικών διαμερίσεων (NP) (βλ. Διάγραμμα 2.1).



Διάγραμμα 2.1: Γειτονικές διαμερίσεις

Μία διαμέριση NP ορίζεται να είναι γειτονική της διαμέρισης P αν και μόνο αν

η NP είναι ακριβώς η ίδια όπως η διαμέριση P, εκτός του ότι ένα μόνο στοιχείο του υποσυστήματος στη διαμέριση P είναι σε ένα διαφορετικό υποσύστημα στη διαμέριση NP. Εάν η διαμέριση P περιέχει κόμβους m και k υποσυστήματα, ο συνολικός αριθμός των γειτόνων είναι $O(n•k)$. Θα πρέπει να σημειωθεί ότι για πολλές διαμερίσεις ενός MDG ο αριθμός των γειτόνων είναι ακριβώς $n•k$. Ωστόσο, αν ένα τμήμα περιέχει υποσυστήματα με 1 ή 2 κόμβους, ο συνολικός αριθμός των διακριτών γειτόνων είναι ελαφρώς μικρότερος.

Αλγόριθμος συσταδοποίησης αναρρίχησης λόφων

Ο Αλγόριθμος 2 αναλύει τη συμπεριφορά του αλγορίθμου αναρρίχησης λόφων και αποτελεί μία προσεγγιστική μετάφραση σε ελληνική ψευδογλώσσα του αντίστοιχου στο Mitchell και Mancoridis 2002a. Η συνάρτηση HillClimbing(...) διαχειρίζεται τις μεμονωμένες διαμερίσεις στον πληθυσμό, και η λειτουργία ClimbHill (...) λαμβάνει μία συγκεκριμένη διαμέριση και τη «βελτιώνει» χρησιμοποιώντας την τεχνική δημιουργίας γειτονικών τμημάτων που περιγράφεται παραπάνω. Η συμπεριφορά του αλγορίθμου που σχετίζεται με κάποια περιοχή στον ψευδοκώδικα εξηγείται στον Πίνακα 2.1. Σε μια προσπάθεια να αποφευχθεί η υπερβολική πολυπλοκότητα στον ψευδοκώδικα του Αλγόριθμου 2, αφέθηκαν σκόπιμα έξω μερικά χαρακτηριστικά του αλγορίθμου αναρρίχησης λόφων. Αυτά τα χαρακτηριστικά του περιγράφονται στη συνέχεια.

2.3.3 Βελτιώσεις στον αλγόριθμο αναρρίχησης λόφων

Η προσέγγιση συσταδοποίησης με αναρρίχηση λόφων, επιχειρεί να λάβει μία διαμέριση και να τη βελτιώσει εξετάζοντας ένα ποσοστό των γειτονικών κατατμήσεων της. Συγκεκριμένα, η `ClimbHill (...)` επιστρέφει την ίδια διαμέριση που έλαβε ως είσοδο για να δείξει τη σύγκλιση, ή μία βελτιωμένη διαμέριση, εάν κάποια μπορεί να βρεθεί. Ένα πολύ γνωστό πρόβλημα των αλγορίθμων αναρρίχησης λόφων είναι ότι ορισμένα σημεία αρχικής εκκίνησης μπορούν να συγκλίνουν σε κακές λύσεις (δηλαδή, τοπικά βέλτιστα). Για την αντιμετώπιση αυτού του προβλήματος, ο αλγόριθμος αναρρίχησης λόφων δεν βασίζεται σε ένα μόνο τυχαίο σημείο εκκίνησης, αλλά αντ' αυτού

Αλγόριθμος 2 Αλγόριθμος Αναρρίχησης Λόφων

```
1:  HillClimbing(MDG ) Ακέραιος popSz; Κατώφλι t
2:      Έστω P ένα σύνολο τυχαίων διαμερίσεων του M με popSz αριθμό στοιχείων
3:      Έστω B το καλύτερο τμήμα του M, με αρχική τιμή το ∅
4:      Έστω max_mq = -∞
5:      Για κάθε διαμέριση p ∈ P επανέλαβε
6:          Έστω currentP = p
7:          Έστω nextP = ClimbHill(currentP, t)
8:          όσο MQ(nextP) > MQ(currentP) επανέλαβε
9:              currentP ← nextP
10:             nextP = ClimbHill(currentP , t)
11:         ΤέλοςΕπανάληψης
12:         Έστω mq_hc = MQ(currentP )
13:         Αν mq_hc > max_mq τότε
14:             B ← currentP
15:             max_mq ← mq_hc
16:         ΤέλοςΑν
17:     ΤέλοςΕπανάληψης
18:     επέστρεψε (διαμέριση B)
19: ClimbHill(Διαμέριση P; Κατώφλι η)
20:     Έστω BestP = P
21:     Έστω neighborEvalCnt = (P.numClusters) × (P.MDG.numNodes) × η
22:     Έστω N το σύνολο γειτώνων του P
23:     N ← randomize(N)
24:     Έστω max_mq = MQ(P)
25:     Έστω count = 0
26:     Έστω improved = false
27:     Για κάθε neighbor n ∈ N επανέλαβε
28:         count ← count + 1
29:         Έστω διαμέριση C = P.applyNeighbor(n)
30:         Έστω mq_n = MQ(C)
31:         Αν mq_n > max_mq τότε
32:             BestP ← C
33:             max_mq ← mq_n
34:             improved = true
35:         ΤέλοςΑν
36:         Αν ( count ≥ neighborEvalCnt KAI improved = true) τότε
37:             επέστρεψε (διαμέριση BestP)
38:         ΤέλοςΑν
39:     ΤέλοςΕπανάληψης
40:     επέστρεψε (διαμέριση BestP)
```

Πίνακας 2.1: Επεξήγηση του Αλγόριθμου Αναρρίχησης Λόφων

Σειρά	**Περιγραφή**
2.2	P είναι ένα σύνολο τυχαίων διαμερίσεων του MDG M. Το μέγεθος του P παρέχεται από την παράμετρο της *HillClimbing()*, $popSz$
2.8	Ο βρόχος επανάληψης **όσο...επανέλαβε**(...) χρησιμοποιεί τη λειτουργία *ClimbHill()* για να βελτιωθεί μία από τις αρχικές τυχαίες διαμερίσεις του M. Κάθε φορά, η διαμέριση που επέστρεψε η λειτουργία *ClimbHill ()* συγκρίνεται με αυτή που προκύπτει από την προηγούμενη επανάληψη. Το τμήμα συγκλίνει όταν η MQ παύει να βελτιώνεται
2.13	Η *HillClimbing()* παρακολουθεί την καλύτερη διαμέριση (που έχει βελτιωθεί με τη χρήση της *ClimbHill()*) του αρχικού πληθυσμού. Σε οποιαδήποτε χρονική στιγμή, το καλύτερο τμήμα αποθηκεύεται στο B. Η τελευταία εντολή στη *HillClimbing()* επιστρέφει το B. Αυτό είναι το καλύτερο υπο-βέλτιστο αποτέλεσμα με βάση τον αρχικό τυχαίο πληθυσμό M.
2.21	Ο αλγόριθμος δέχεται μία παράμετρο κατωφλιού η που αντιπροσωπεύει τον ελάχιστο αριθμό των γειτόνων που πρέπει να εξεταστούν κατά τη διάρκεια της διαδικασίας του. Εάν η = 0%, τότε ο πρώτος βελτιωμένος γείτονας θα επιστραφεί. Εάν η = 100%, τότε όλοι οι γείτονες θα εξεταστούν και θα επιστραφεί ο γείτονας που παράγει τη μεγαλύτερη βελτίωση στην MQ. Για ενδιάμεσες τιμές του η $(0\% \leq \eta \leq 100\%)$ το κατώφλι χρησιμοποιείται για τον υπολογισμό neighborEvalCnt, η οποία υπολογίζει τον αριθμό ελάχιστων γειτόνων που πρέπει να εξεταστούν. Αν δεν βρεθεί βελτιωμένος γείτονας πριν εξεταστούν οι neighborEvalCnt γείτονες, συνεχίζει να ψάχνει για ένα καλύτερο γείτονα. Ο πρώτος βελτιωμένος γείτονας που βρίσκεται πέρα των neighborEvalCnt επιστρέφεται. Αν όλοι οι γείτονες της P εξεταστούν, και αν δεν βρεθεί βελτιωμένος γείτονας, τότε η *ClimbHill()* συγκλίνει.
2.23	Οι γείτονες του εξετάζονται με τυχαία σειρά. N είναι το σύνολο των τυχαία τοποθετημένων γειτόνων.
2.27	Ο βρόγχος επαναλαμβάνεται για όλους τους γείτονες στο N.
2.29	Το τμήμα C είναι ένας γείτονας του P και προσδιορίζεται με την εφαρμογή γειτονική κίνησης n ως προς το P
2.31	Η **Αν**(...) ελέγχει αν το γειτονικό τμήμα C διαθέτει *MQ* καλύτερη από οποιαδήποτε έχει δει μέχρι τώρα. Αν ναι, το τμήμα *C* αποθηκεύεται στη μεταβλητή BestP, και η σημαία improved ορίζεται σε true.
2.36	Η **Αν**(...) ελέγχει αν έχουμε εξετάσει τον απαιτούμενο αριθμό γειτόνων. Αν ναι, και η βελτιωμένη σημαία είναι true, τότε η BestP επιστρέφεται. Αν ο απαιτούμενος αριθμός των γειτόνων δεν έχει ακόμη εξετασθεί, τότε η *ClimbHill ()* συνεχίζει την αναζήτηση.

χρησιμοποιεί μια συλλογή από τυχαία σημεία εκκίνησης. Σε αυτήν την ενότητα περιγράφουμε μερικές ακόμα βελτιώσεις που έγιναν στο παραδοσιακό αλγόριθμο αναρρίχησης λόφων.

Προσομοιωμένη ανόπτηση (Simulated Annealing)

Ένας άλλος τρόπος για να ξεπεραστεί το ανωτέρω περιγραφέν πρόβλημα είναι να χρησιμοποιηθεί η τεχνική προσομοιωμένης ανόπτησης. Όταν εφαρμόζεται σε προβλήματα βελτιστοποίησης, επιτρέπει την αναζήτηση να δεχθεί, με κάποια πιθανότητα, μια χειρότερη παραλλαγή ως νέα λύση της τρέχουσας επανάληψης. Καθώς οι υπολογισμοί προχωράν, η πιθανότητα μειώνεται. Όσο πιο αργό το πρόγραμμα ψύξης, ή ρυθμός μείωσης, τόσο πιο πιθανό είναι ο αλγόριθμος να βρει μια βέλτιστη ή σχεδόν βέλτιστη λύση. Οι Simulated Annealing τεχνικές τυπικά αντιπροσωπεύουν το πρόγραμμα σε λειτουργία ψύξης που μειώνει την πιθανότητα αποδοχής χειρότερης παραλλαγής στα στάδια βελτιστοποίησης του αλγόριθμου.

Στη συνάρτηση `ClimbHill (...)` για το λόγο αυτό ορίζεται μια λειτουργία ψύξης που καθορίζει την πιθανότητα αποδοχής μιας χειρότερης λύσης. Η ιδέα είναι ότι με αποδοχή ενός χειρότερου γείτονα, περιστασιακά ο αλγόριθμος θα κάνει ένα ”άλμα” για να διερευνήσει μια νέα περιοχή στο χώρο αναζήτησης.

Η λειτουργία ψύξης έχει σχεδιαστεί για να σεβαστεί τις ιδιότητες του χρονοδιαγράμματος ψύξης: (α) μειώνει την πιθανότητα αποδοχής μιας χειρότερης κίνησης με την πάροδο του χρόνου, και (β) αυξάνει την πιθανότητα της αποδοχής μιας χειρότερης κίνησης, αν ο ρυθμός βελτίωσης είναι μικρός. Παρακάτω παρουσιάζεται η λειτουργία ψύξης όπως έχει σχεδιαστεί σε σχέση με τις παραπάνω απαιτήσεις.

$$P(A) = \begin{cases} 0 & \triangle MQ \geq 0 \\ e^{\frac{\triangle MQ}{T}} & \triangle MQ < 0 \end{cases}$$

$$T(k+1) = \alpha \bullet T(k)$$

Κάθε φορά που η πιθανότητα υπολογίζεται, το $T(\kappa)$ μειώνεται. Η αρχική τιμή του T (δηλαδή,$T(0)$) και η σταθερά του ποσοστού μείωσης καθορίζονται από το χρήστη. Επιπλέον, η MQ πρέπει να είναι αρνητική, γεγονός που σημαίνει ότι η τιμή της MQ έχει μειωθεί. Μόλις η πιθανότητα υπολογίζεται, επιλέγεται ένας ομοιόμορφος τυχαίος

αριθμός μεταξύ του 0 και του 1. Εάν αυτός ο τυχαίος αριθμός είναι μικρότερος από την πιθανότητα $P(A)$, το τμήμα είναι αποδεκτό.

Βαθμονόμηση του Αλγόριθμου Ομαδοποίησης

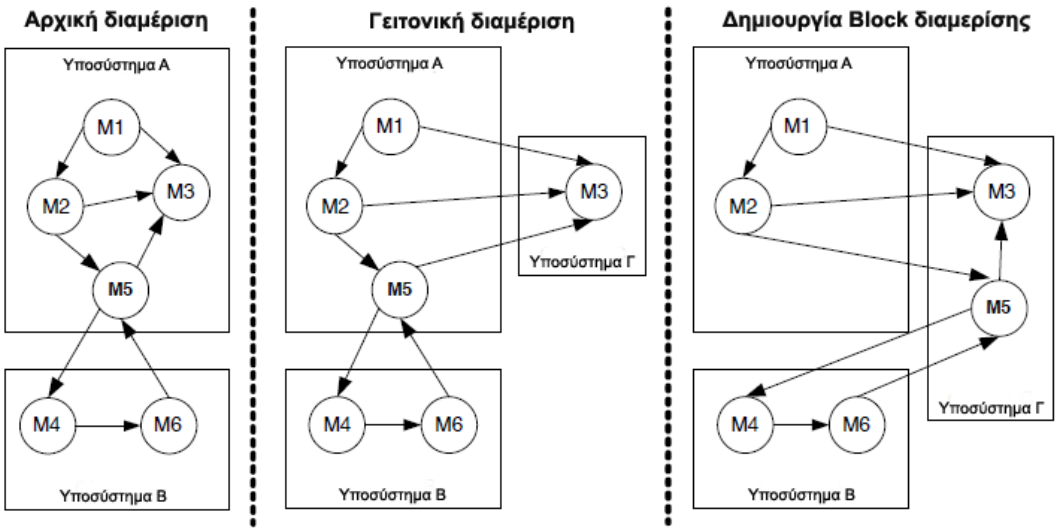
Ο αλγόριθμος αναρρίχησης λόφων χρησιμοποιεί ένα κατώτατο όριο (κατώφλι) η $(0\% \leq \eta \leq 100\%)$ για τον υπολογισμό του ελάχιστου αριθμού γειτόνων που πρέπει να εξεταστούν κατά τη διάρκεια κάθε επανάληψης της διαδικασίας αναρρίχησης λόφων. Μια χαμηλή τιμή για η έχει ως αποτελέσματα ο αλγόριθμος να κάνει περισσότερα μικρά βήματα πριν να συγκλίνει, ενώ μια υψηλή τιμή για η έχει αποτέλεσμα ο αλγόριθμος να κάνει λιγότερα μεγάλα βήματα πριν να συγκλίνει. Εμπειρικά έχει δειχθεί ότι εξετάζοντας πολλούς γείτονες κατά τη διάρκεια κάθε επανάληψης (δηλαδή, χρησιμοποιώντας ένα μεγάλο κατώφλι, όπως $\eta \geq 75\%$) αυξάνεται ο χρόνος σύγκλισης του αλγορίθμου σε ένα αποτέλεσμα.

Η αύξηση σε χρόνο εκτέλεσης, όμως, έχει ως αποτέλεσμα ο αλγόριθμος να διερευνήσει περισσότερο το χώρος αναζήτησης πριν από την επιλογή ενός τμήματος του MDG για να βρει τη βάση της επόμενης επανάληψης αναζήτησης. Εξετάζοντας βαθύτερα το χώρου αναζήτησης αυξάνεται η πιθανότητα να βρούμε μία καλύτερη λύση. Στην υλοποίηση του αλγόριθμο αναρρίχησης λόφων, το κατώφλι η μπορεί να ρυθμιστεί από τη διεπαφή του χρήστη ή προγραμματιστικά μέσω API.

Δημιουργία Block διαμερίσεων

Όταν αξιολογείται το σύνολο των γειτόνων για την τρέχουσα διαμέριση, κάθε κόμβος στο MDG κινείται όχι μόνο προς τα υπόλοιπα υποσυστήματα στη διαμέριση, αλλά επίσης σε ένα νέο μονήρες υποσύστημα. Επειδή η MQ προσπαθεί να ελαχιστοποιήσει συζεύξεις, είναι σπάνιο ότι η εισαγωγή ενός μονήρους υποσυστήματος θα δώσει ως αποτέλεσμα μια βελτιωμένη τιμή της MQ. Αυτό το χαρακτηριστικό της MQ εισάγει μια ανεπιθύμητη παρενέργεια, δηλαδή, ο αλγόριθμος αναρρίχησης λόφων θεωρεί τη δημιουργία πρόσθετων υποσυστημάτων ως δυσμενή εξέλιξη. Για την αντιμετώπιση αυτού του προβλήματος, αναθεωρήθηκε ελαφρώς ο ορισμός για το τι είναι ένα γειτονικό τμήμα. Κατά τη διαδικασία αναρρίχησης λόφων, εκτός από την εξέταση των γειτονικών τμημάτων, εξετάζεται επίσης ένα νέο είδος διαμέρισης, που ονομάζεται block διαμέριση. Ορίζεται διαμέριση BBP (βλ. Διάγραμμα 2.2) να είναι μια block διαμέριση

του P αν και μόνο αν η BBP είναι ακριβώς η ίδια με τη P εκτός από το ότι: (α) BBP περιέχει ένα επιπλέον υποσύστημα, και (β) το νέο υποσύστημα περιέχει ακριβώς δύο κόμβους από τον MDG.



Διάγραμμα 2.2: Δημιουργία Block διαμερίσεων

Τεχνικά, υπάρχουν $|V|^2$ block διαμερίσεις αν ταιριάξουμε κάθε στοιχείο με όλα τα άλλα στοιχεία κατά τη δημιουργία μίας block διαμέρισης. Ωστόσο, όταν ένας κόμβος κινείται σε ένα μονήρες υποσύστημα, δημιουργούμε μόνο block διαμερίσεις με άλλους κόμβους που συνδέονται με αυτόν τον κόμβο από μία ακμή (σχέση) στον MDG. Αυτή η βελτιστοποίηση μειώνει τον συνολικό αριθμό των block διαμερίσεων από $O(|V|^2)$ έως $O(|E|)$.

2.3.4 Προσαρμοσμένος Αλγόριθμος Αναρρίχησης Λόφων

Όπως αναφέρθηκε νωρίτερα ο χώρος αναζήτησης αυξάνεται με γεωμετρική πρόοδο σε σχέση με το σύνολο των κόμβων, που έχει ως αποτέλεσμα τη ραγδαία επιβράδυνση του αλγόριθμου σε εκθετική αναλογία με την αύξηση του συνόλου. Αυτό καθιστά τον αλγόριθμο αναρρίχησης λόφων κατάλληλο και αποτελεσματικό για γράφους με περιορισμένο αριθμό κόμβων, ενώ σε αντίθετη περίπτωση τείνει να είναι αργός και πέρα από ορισμένες τιμές να αδυνατεί να καταλήξει σε αποτέλεσμα μέσα σε λογικά χρονικά πλαίσια.

Κατά παραδοχή το λογισμικό συσταδοποίησης που παρουσιάστηκε έχει νόημα και είναι ωφέλιμο σε γράφους με μεγάλο πληθυσμό κόμβων. Για να διορθωθεί λοιπόν το

παραπάνω πρόβλημα εφαρμόστηκαν μερικές τροποποιήσεις στον αρχικό αλγορίθμο ώστε αυτός να συγκλίνει πιο γρήγορα.

Ο αρχικός αλγόριθμος σε κάθε επανάληψη του αφού υπολογίσει όλες τις πιθανές γειτονικές διαμερίσεις για όλο το σύνολο των κόμβων του επιλέγει εκείνη με την υψηλότερη MQ και συνεχίζει στην επόμενη επανάληψη. Για παράδειγμα σε ένα γράφο με V κόμβους και E το σύνολο των ακμών, για την απόφαση κάθε μιας κίνησης θα πρέπει να υπολογισθούν τουλάχιστον $O(|E|)$ πιθανές διαμερίσεις. Διαμορφώνοντας τον αλγόριθμο να επιλέγει την επόμενη διαμέριση κατά την εξέταση των γειτονικών διαμερίσεων ανά κόμβο, του επιτρέπουμε μέχρι και V κινήσεις σε μία επανάληψη. Σύμφωνα με τις παραπάνω τροποποιήσεις ο Αλγόριθμος 2 μετασχηματίζεται στον Αλγόριθμο 3 που περιγράφεται παρακάτω.

Παρατηρούμε ότι από τον Αλγόριθμο 3 δεν υπάρχει η δυνατότητα επιλογής κατωφλιού αφού αυτό δε συμβάλει πλέον στη μείωση του χρόνου εκτέλεσης, αλλά αντίθετα επηρεάζει αρνητικά την ικανότητα του αλγόριθμου να συγκλίνει στο βέλτιστο αποτέλεσμα.

Επίσης από τον προσαρμοσμένο αλγόριθμο αναρρίχησης λόφων αφαιρέθηκε και η προσομοιωμένη ανόπτηση, όπως περιγράφεται παραπάνω. Η πιθανότητα να δεχθούμε μία χειρότερη διαμέριση ως νέα λύση στη τρέχουσα επανάληψη αποτελεί πλέον την βάση του αλγορίθμου αφού σε κάθε βήμα της επανάληψης δεχόμαστε ως ισχύουσα διαμέριση, τη βέλτιστη διαμέριση από το υποσύνολο των γειτονικών διαμερίσεων του κόμβου που είναι υπό εξέταση και όχι όλων των πιθανών γειτονικών διαμερίσεων. Δηλαδή, αν ο αρχικός αλγόριθμος εξέταζε όλο το πλήθος των γειτονικών διαμερίσεων $O(|E|)$ για να αποφασίσει τη λύση στη τρέχουσα επανάληψη, πλέον παίρνει μία απόφαση ανά $O(|e_v|)$ γειτονικές διαμερίσεις, για κάθε κόμβο $v \in V$ με ακμές e_v, ως ότου εξετασθούν όλοι οι κόμβοι και να τροφοδοτήσει την επόμενη επανάληψη με τη διαμέριση που προέκυψε.

Αλγόριθμος 3 Προσαρμοσμένος Αλγόριθμος Αναρρίχησης Λόφων

HillClimbing(MDG) Ακέραιος popSz
Έστω P ένα σύνολο τυχαίων διαμερίσεων του M με popSz αριθμό στοιχείων
Έστω B το καλύτερο τμήμα του M, με αρχική τιμή το ∅
Έστω max_{mq} = -∞
Για κάθε διαμέριση p ∈ P **επανέλαβε**
Έστω currentP = p
Έστω nextP = **ClimbHill**(currentP)
όσο MQnextP > MQ(currentP) **επανέλαβε**
currentP ← nextP
nextP = ClimbHill(currentP)
ΤέλοςΕπανάληψης
Έστω mq_{hc} = MQ(currentP)
Αν $mq_{hc} > max_{mq}$ **τότε**
B ← currentP
$max_{mq} \leftarrow mq_{hc}$
ΤέλοςΑν
ΤέλοςΕπανάληψης
επέστρεψε (διαμέριση B)
ClimbHill(Διαμέριση P)
Έστω V το σύνολο κόμβων του P
Έστω max_{mq} = MQ(P)
Για κάθε κόμβο v ∈ V **επανέλαβε**
Έστω BestP = P
Έστω N το σύνολο γειτόνων του P για κίνηση του κόμβου v
Για κάθε neighbor n ∈ N **επανέλαβε**
Έστω διαμέριση C = P.applyNeighbor(n)
Έστω mq_n = MQ(C)
Αν $mq_n > max_{mq}$ **τότε**
BestP ← C
$max_{mq} \leftarrow mq_n$
ΤέλοςΑν
ΤέλοςΕπανάληψης
Αν max_{mq} > MQ(P) **τότε**
P ← BestP
ΤέλοςΑν
ΤέλοςΕπανάληψης
επέστρεψε (διαμέριση P)

Κεφάλαιο 3

Ο σχεδιασμός του εργαλείου Sheaf

Σε αυτό το κεφάλαιο παρουσιάζουμε το σχεδιασμό του εργαλείου συσταδοποίησης Sheaf που αναπτύξαμε. Στόχος μας, κατά την επιλογή των εργαλείων που χρησιμοποιήσαμε για την υλοποίηση του, ήταν να πληρούνται οι παρακάτω απαιτήσεις :

- Ευελιξία: Να μπορεί ο πηγαίος κώδικας για το εργαλείο μας να αλλάζει συχνά, το Sheaf είναι ένα εργαλείο που υποστηρίζει την έρευνα, πρέπει να δίνει την ικανότητα να ενσωματωθούν αλλαγές στον πηγαίο κώδικα γρήγορα.

- Εγκατάσταση & φορητότητα: Είναι σημαντικό το εργαλείο μας ναι είναι εύκολο να εγκατασταθεί και να μπορεί να εκτελεστεί σε μια ποικιλία λειτουργικών συστημάτων.

- Απόδοση: Η ταχύτητα εκτέλεσης είναι πολύ σημαντική για να μπορεί να χρησιμοποιηθεί το εργαλείο μας, ώστε να είναι ικανό να συσταδοποιήσει μεγάλα συστήματα.

Η περιγραφή των παραπάνω απαιτήσεων συνέπεσε με την υιοθέτηση της γλώσσας προγραμματισμού Java. Η επιλογή χρήσης μόνο της Java ικανοποίησε πολλές από τις αρχικές απαιτήσεις μας. Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού με μια ισχυρή βιβλιοθήκη επαναχρησιμοποιήσιμων κλάσεων. Η Java είναι ανεξάρτητη από τη πλατφόρμα επειδή ο μεταγλωττιστής της παράγει ενδιάμεσο bytecode που ερμηνεύεται από ένα virtual machine που εκτελείται στο εγγενές λειτουργικό σύστημα. Με γνώμονα αυτές τις δυνατότητες, υλοποιήσαμε την αρχική έκδοση του Sheaf σε Java.

Για να εκτελέσουμε την διαδικασία εισόδου/εξόδου και οπτικοποίησης χρησιμοποιήσαμε διάφορα εργαλεία με το Sheaf. Σημαντικότερο από αυτά το API της Gephi. Το Gephi είναι ένα πακέτο λογισμικού ανοιχτού κώδικα για την ανάλυση και την απεικόνιση γράφων και είναι γραμμένο σε Java στην πλατφόρμα NetBeans.

Εκτός λοιπόν από την οπτικοποίηση του γράφου, οι βιβλιοθήκες της Gephi προσθέσαν και ένα εύρος ακόμα δυνατοτήτων στο εργαλείο Sheaf. Αρχικά απλουστεύσανε τη διαδικασία αποθήκευσης και εξαγωγής αρχείων , δίνοντας μας την επιλογή να διαλέξουμε ανάμεσα σε έξι διαφορετικούς τύπους αρχείων γράφων και τρεις εικόνας. Επίσης καθιστούν ικανό το άνοιγμα οποιαδήποτε αρχείου γράφου από τα παραπάνω. Τέλος οι υπηρεσίες οπτικοποίησης, εμφάνισης, διάταξης και φιλτραρίσματος που αναφέρουμε αναλυτικότερα σε παρακάτω κεφάλαιο, υλοποιούνται με τη βοήθεια αυτών των βιβλιοθηκών.

Άλλα εργαλεία που χρησιμοποιήσαμε είναι τα παρακάτω:

Maven Το Maven είναι ένα εργαλείο αυτοματοποίησης δημιουργίας προγραμμάτων που χρησιμοποιείται κυρίως στη Java. Το Maven εξετάζει δύο πτυχές του λογισμικού κατασκευής: πρώτον, περιγράφει τον τρόπο κατασκευής του λογισμικού και, δεύτερον, περιγράφει τις εξαρτήσεις του.

opencsv Το opencsv είναι μια εύκολη στη χρήση βιβλιοθήκη ανάλυσης CSV (χωρισμένα με κόμματα) για την Java.

JUnit Το JUnit είναι ένα unit testing framework για τη γλώσσα προγραμματισμού Java. Το JUnit ήταν σημαντικό στην test-driven development και ανήκει στην οικογένεια των unit testing framework που είναι γνωστά ως xUnit και προέρχεται από το SUnit.

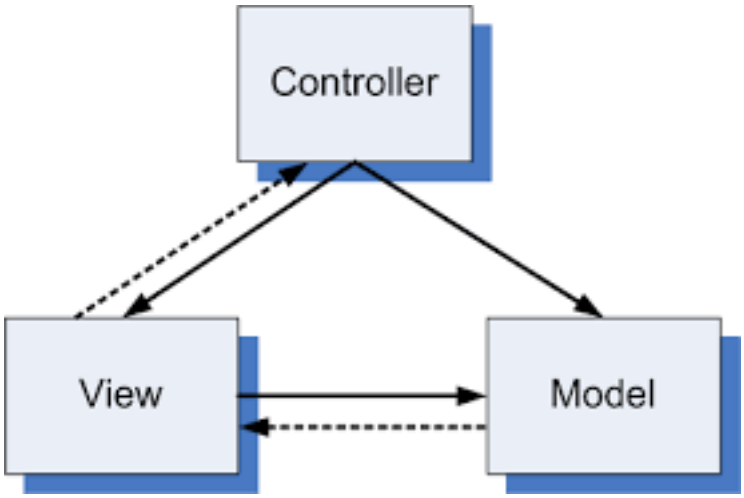
Synthetica & Synthetica Simple2D Το Synthetica είναι ένα Look and Feel[1] που διανέμεται από την εταιρία Jyloo Software. Το 'Simple2D Look and Feel' δεν μπορεί να χρησιμοποιηθεί αυτόνομα (χωρίς το Synthetica).

JFontChooser Ένα Java component ανοιχτού κώδικα για την επιλογή γραμματοσειράς.

[1]Look and Feel είναι ένας όρος που χρησιμοποιείται σε σχέση με ένα γραφικό περιβάλλον χρήστη και περιλαμβάνει πτυχές του σχεδιασμού του, συμπεριλαμβανομένων στοιχείων όπως τα χρώματα, τα σχήματα, τη διάταξη και τις γραμματοσειρές (Look), καθώς και τη συμπεριφορά δυναμικών στοιχείων όπως κουμπιά, κουτιά και μενού (Feel).

3.1 Αρχιτεκτονική του Sheaf

Η υλοποίηση του εργαλείου Sheaf ακολουθεί το αρχιτεκτονικό μοντέλο λογισμικού **Model–View–Controller (MVC)**. Το MVC είναι ένα αρχιτεκτονικό μοντέλο λογισμικού για την υλοποίηση διεπαφών χρήστη σε υπολογιστές. Διαχωρίζει μια συγκεκριμένη εφαρμογή σε τρία διασυνδεδεμένα μέρη. Αυτό γίνεται για να διαχωριστούν οι εσωτερικές παραστάσεις πληροφοριών από τους τρόπους με τους οποίους οι πληροφορίες παρουσιάζονται και γίνονται δεκτές από τον χρήστη. Το μοτίβο σχεδίασης MVC αποσυνδέει αυτά τα κύρια στοιχεία επιτρέποντας την αποτελεσματική επαναχρησιμοποίηση κώδικα και την παράλληλη ανάπτυξη. Η αποσύνθεση υποσυστημάτων του Sheaf χρησιμοποιήθηκε επίσης για την οργάνωση του πηγαίου κώδικα. Κάθε αρχείο πηγαίου κώδικα τοποθετείται στο πακέτο Java που αντιστοιχεί στο υποσύστημα του.



Διάγραμμα 3.1: Το μοντέλο MVC.

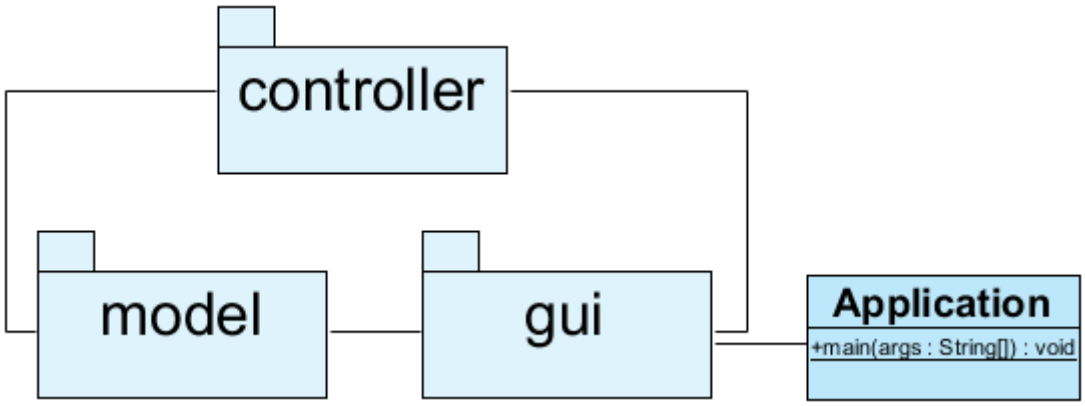
Στο διάγραμμα 3.1 βλέπουμε τη χαρακτηριστική δομή του MVC ενώ στο 3.2 την αντίστοιχη δική μας. Παρακάτω αναφέρουμε αναλυτικότερα τα κύρια στοιχεία του MVC:

- Το **model** είναι το κεντρικό στοιχείο του σχεδίου. Εκφράζει τη συμπεριφορά της εφαρμογής από την άποψη του τομέα προβλημάτων, ανεξάρτητα από το περιβάλλον χρήστη. Διαχειρίζεται άμεσα τα δεδομένα, τη λογική και τους κανόνες της εφαρμογής.

- Το **view** μπορεί να είναι οποιαδήποτε αναπαράσταση εξόδου πληροφοριών, όπως

ένα γράφημα ή ένα διάγραμμα. Είναι δυνατές πολλαπλές προβολές των ίδιων πληροφοριών, όπως ένα διάγραμμα ράβδων για διαχείριση και μια προβολή σε πίνακες για τους λογιστές.

- Το τρίτο μέρος, ο **controller**, δέχεται είσοδο και τη μετατρέπει σε εντολές για το model ή το view.



Διάγραμμα 3.2: Η αρχιτεκτονική του Sheaf.

Εκτός από τη διαίρεση της εφαρμογής σε τρία είδη στοιχείων, ο σχεδιασμός του model-view-controller ορίζει τις αλληλεπιδράσεις μεταξύ τους.

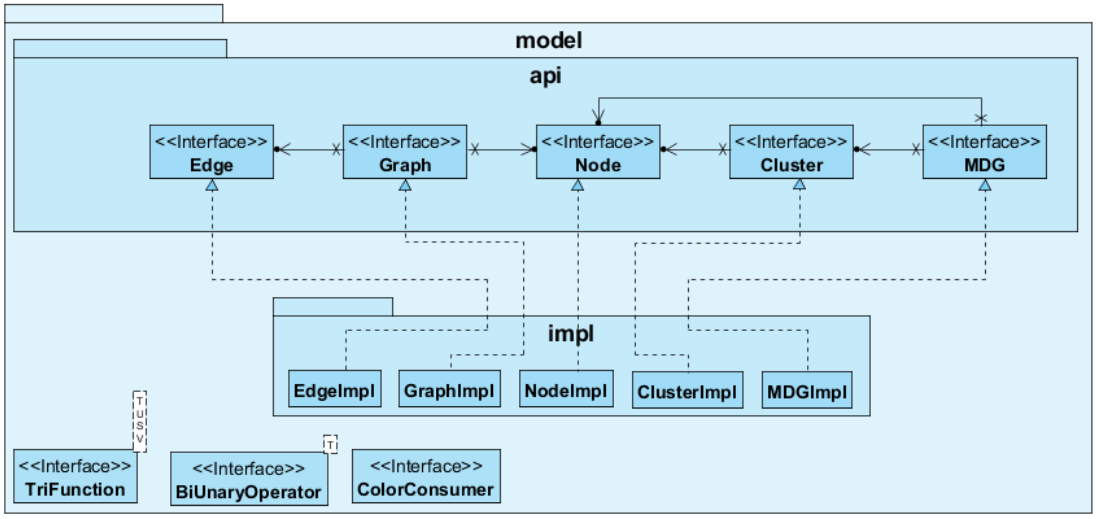
- Ένα model αποθηκεύει δεδομένα που ανακτώνται σύμφωνα με εντολές από τον controller και εμφανίζονται στο view.
- Ένα view παράγει νέα αποτελέσματα στον χρήστη βάσει αλλαγών στο model.
- Ένας controller μπορεί να στείλει εντολές στο model για να ενημερώσει την κατάσταση του model. Μπορεί επίσης να στείλει εντολές στο σχετικό view για να αλλάξει την παρουσίαση του view του model.

3.2 Υποσυστήματα του Sheaf

Σε αυτό το κεφάλαιο θα ασχοληθούμε με τα επιμέρους υποσυστήματα του εργαλείου Sheaf. Όπως αναφέραμε παραπάνω το σύστημα μας αποσυντίθεται σε τρία κύρια στοιχεία, **model**, **controller** και **view**. Θα αναλύσουμε το καθένα ξεχωριστά περιγράφοντας τη δομή του, τα επιμέρους στοιχεία του, την εσωτερική τους συνδεσιμότητα αλλά και την αλληλεπίδραση τους με τα άλλα υποσυστήματα.

3.2.1 Υποσύστημα model

Στο διάγραμμα 3.3 βλέπουμε την εσωτερική δομή του υποσυστήματος model. Το υποσύστημα model αποτελείται από δύο φακέλους, τον api που περιέχει τις γενικευμένες δομές που χρησιμοποιούμε για τη διαδικασία συσταδοποίησης και το impl, ο οποίος περιέχει τις υλοποιήσεις τους. Επίσης έχουμε μερικά functional interfaces για την απλούστευση κάποιων διαδικασιών.



Διάγραμμα 3.3: Το υποσύστημα model

Οι δύο αρχικές δομές μας είναι το **Node** και **Edge**, που αντιπροσωπεύουν κόμβους και ακμές αντίστοιχα. Οι απαραίτητες ιδιότητες για την υλοποίηση ενός κόμβου είναι ένας μοναδικός ακέραιος αριθμός (*id*), το όνομα το εκάστοτε κόμβου (*name*) και ένας χάρτης που αντιστοιχεί τους γειτονικούς κόμβους με τη τιμή του βάρους της ακμής που τους συνδέει (*neighbours*). Ενώ οι ιδιότητες των ακμών είναι το id του πηγαίου κόμβου (*sourceID*), το id του στόχου κόμβου (*targetID*), ο τύπος ακμής (*type*) και το το βάρος της (*weight*).

Παρακάτω παραθέτουμε το κώδικα και των δύο δομών που περιέχει τις συναρτήσεις που υλοποιούν οι κλάσεις `NodeImpl` και `EdgeImpl` αντίστοιχα.

```
public interface Node {
    int getId();
    String getName();
    HashMap<Node, Float> getNeighbours();
    void addNeighbour(Node neighbour, float weight);
}
```

```
public interface Edge {
    int getSourceID();
    void setSourceID(int sourceID);
    int getTargetID();
    void setTargetID(int targetID);
    String getType();
    void setType(String type);
    float getWeight();
    void setWeight(float weight);
}
```

Η δομή **Graph** αποτελείται από μία συλλογή κόμβων (*nodes*) και μία ακμών (*edges*). Χρησιμοποιείται μόνο για τη συλλογή όλων των στοιχείων που παίρνουμε ως είσοδο σε ένα αντικείμενο.

```
public interface Graph {
    Set<Node> getNodes();
    void setNodes(Set<Node> nodes);
    Set<Edge> getEdges();
    void setEdges(Set<Edge> edges);
}
```

Περισσότερο ενδιαφέρον παρουσιάζει η δομή **Cluster** που αναπαριστά ένα υποσύστημα του γράφου. Περιέχει πληροφορίες για τους κόμβους που το απαρτίζουν, αλλά επίσης υπολογίζει και το συντελεστή υποσυστήματος που χρησιμοποιείται στην διαδικασία συσταδοποίησης του γράφου βάσει του πλήθους των εσωτερικών και εξωτερικών ακμών του υποσυστήματος. Στη δική μας υλοποίηση το πλήθος των ακμών (εσωτερικών και εξωτερικών) ενημερώνεται αυτόματα κατά τη διαδικασία πρόσθεσης ή αφαίρεσης κόμβου από το υποσύστημα και στη συνέχεια υπολογίζεται ο συντελεστής υποσυστήματος (CF) σύμφωνα με τον τύπο που αναφέρθηκε στην ενότητα 2.2. Οι ιδιότητες που προϋποθέτει η υλοποίηση της είναι ένα αλφαριθμητικό ταυτοποίησης του υποσυστήματος (*id*), μια συλλογή κόμβων (*nodes*), το πλήθος εσωτερικών ακμών του υποσυστήματος (*intraedges*), το πλήθος εξωτερικών ακμών του υποσυστήματος (*interedges*), και το συντελεστή υποσυστήματος (*clusterFactor*).

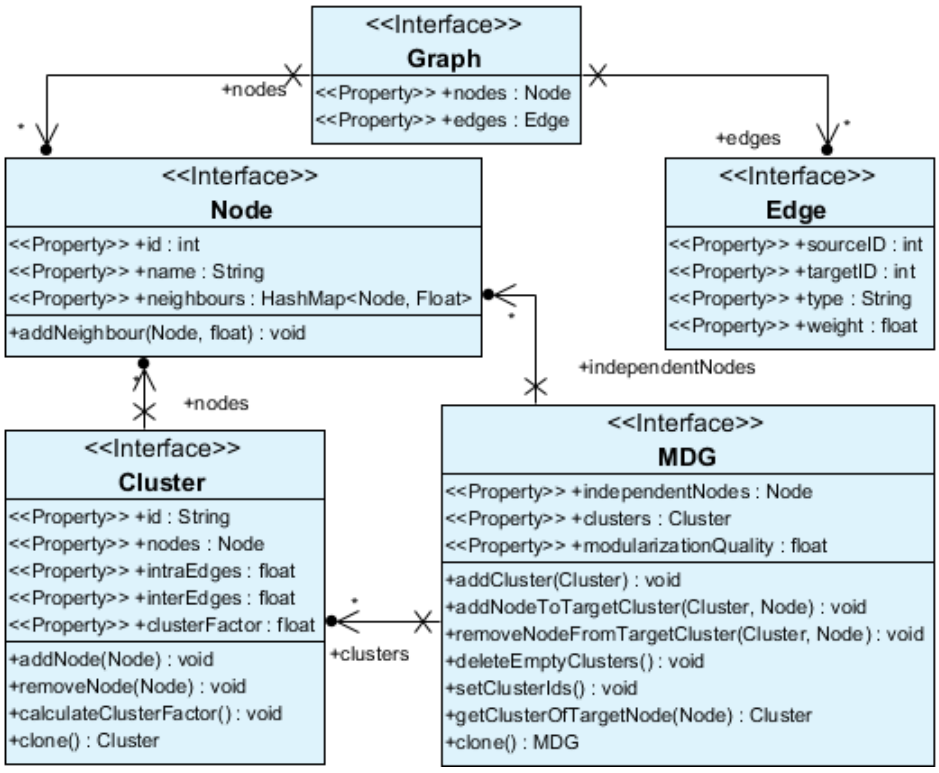
```
public interface Cluster {
    String getId();
    void setId(String id);
    Set<Node> getNodes();
    void setNodes(Set<Node> nodes);
    void addNode(Node node);
    void removeNode(Node node);
    void setIntraEdges(float intraEdges);
    void setInterEdges(float interEdges);
    float getIntraEdges();
    float getInterEdges();
    float getClusterFactor();
    void setClusterFactor(float clusterFactor);
    void calculateClusterFactor();
    Cluster clone();
}
```

Τέλος, η δομή **MDG** απαρτίζεται από ένα σύνολο κόμβων (*independentNodes*) που αντιπροσωπεύουν τους κόμβους του γράφου που δεν παίρνουν μέρος σε καμία εξάρτηση(ακμή) με άλλους, ένα σύνολο υποσυστημάτων *(clusters*) και την ποιότητα κατάτμησης (*modularizationQuality*). Χρησιμοποιείται κατά τη διαδικασία συσταδοποίησης και αντιπροσωπεύει το διαμερισμένο σε υποσυστήματα γράφο. Απομονώνουμε τους ανεξάρτητους κόμβους από τα υποσυστήματα για να διευκολύνουμε τη διαδικασία συσταδοποίησης ελαφρύνοντας το φορτίο ταξινόμησης κόμβων εφόσον η κίνηση το εν λόγω κόμβων ανάμεσα σε υποσυστήματα δε συμβάλει στην αύξηση της MQ. Η MQ υπολογίζεται αυτόματα κατά τη δική μας υλοποίηση κατά την πρόσθεση ή αφαίρεση υποσυστημάτων ή κόμβων σε αυτά.

```
public interface MDG {
    Set<Node> getIndependentNodes();
    Set<Cluster> getClusters();
    void setIndependentNodes(Set<Node> independentNodes);
    void addCluster(Cluster cluster);
    float getModularizationQuality();
```

```
    void setModularizationQuality(float modularizationQuality);
    void addNodeToTargetCluster(Cluster cluster, Node node);
    void removeNodeFromTargetCluster(Cluster cluster, Node
       node);
    void deleteEmptyClusters();
    void setClusterIds();
    Cluster getClusterOfTargetNode(Node node);
    MDG clone();
}
```

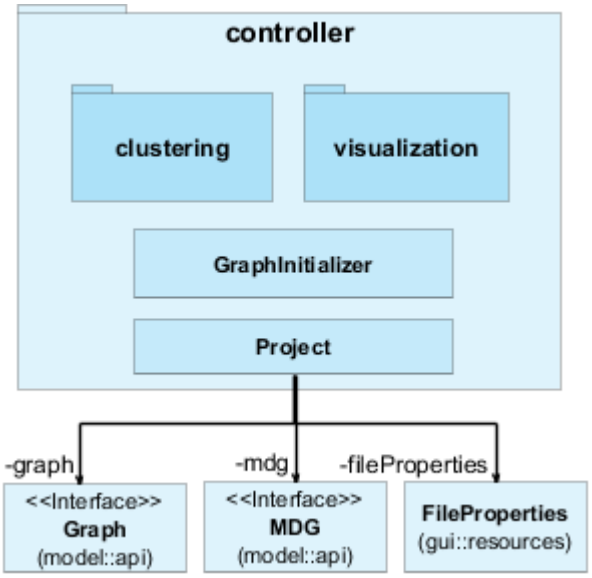
Οι σχέσεις ανάμεσα στις δομές που μόλις περιγράψαμε φαίνονται καλύτερα στο διάγραμμα 3.4. Παρατηρούμε ότι η κεντρική δομή είναι η `Node`, αυτό είναι λογικό εφόσον οι κόμβοι είναι το κύριο στοιχείο των γράφων και η χρήση των ακμών έχει αντικατασταθεί και αυτή από τη `Node` που πλέον περιέχει τις χρήσιμες πληροφορίες της `Edge` για την διαδικασία συσταδοποίησης, στην ιδιότητα της *neighbours*. Είναι φανερό ότι οι δομές `Graph` και `Edge` είναι απομονωμένες από τις δομές που σχετίζονται με τη διαδικασία συσταδοποίησης και χρησιμοποιούνται μόνο για αποθήκευση των στοιχείων και την οπτικοποίηση του γράφου.



Διάγραμμα 3.4: API

3.2.2 Υποσύστημα controller

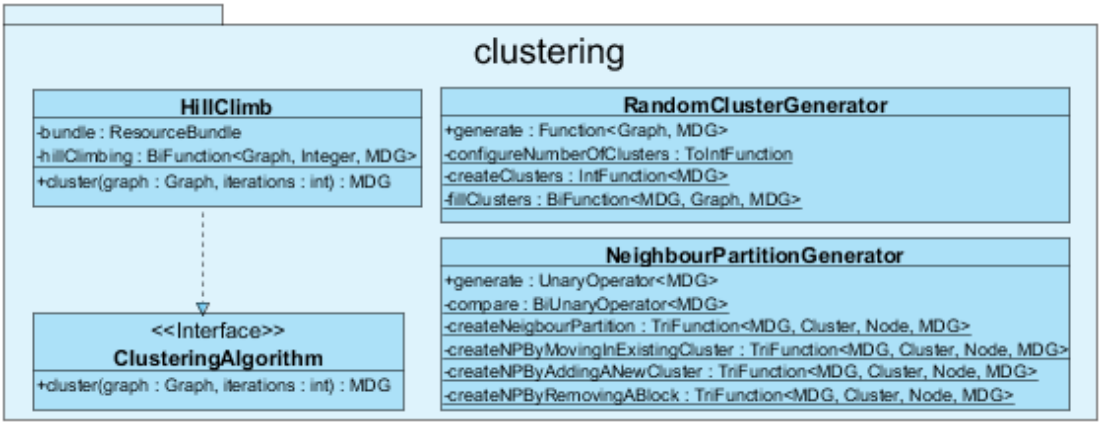
Στο υποσύστημα του model είδαμε τις δομές που χρησιμοποιούμε για την αποθήκευση και τη συσταδοποίηση του γράφου. Για την οπτικοποίηση του χρησιμοποιούμε έτοιμες δομές της βιβλιοθήκης Gephi. Το υποσύστημα του controller περιέχει αρχεία τα οποία βοηθούν στην επικοινωνία των παραπάνω δομών και του view (gui). Ουσιαστικά ο controller στέλνει εντολές στο μοντέλο και ενημερώνει την κατάσταση του. Μπορεί επίσης να στέλνει εντολές ώστε να γίνει η αντίστοιχη αναπαράσταση των δεδομένων του μοντέλου μέσω του view.



Διάγραμμα 3.5: Το υποσύστημα του controller

Στο διάγραμμα 3.5 βλέπουμε την γενικευμένη δομή του υποσυστήματος του controller. Παρατηρούμε τούς δύο κύριους φακέλους **clustering** και **visualization** που περιέχουν αρχεία σχετικά με τη συσταδοποίηση και οπτικοποίηση του γράφου αντίστοιχα. Επίσης έχουμε δύο ακόμα αρχεία το **Project** και το **GraphInitializer**. Το **Project** λειτουργεί ως ο κεντρικός controller για κάθε νέο γράφο που δημιουργούμε και κρατάει πληροφορίες για τον αρχικό γράφο (Graph), το συσταδοποιημένο γράφο (MDG) και το όνομα, θέση και τύπο αρχείου από όπου ανοίχτηκε ή / και αποθηκεύτηκε. Το **GraphInitializer** είναι περισσότερο εργαλείο που χρησιμοποιείται για να μετατρέψει την είσοδο της εφαρμογής στον αντίστοιχο γράφο.

Ο φάκελος clustering περιέχει τα αρχεία, τα οποία είναι υπεύθυνα για τη συσταδοποίηση του γράφου. Όπως βλέπουμε και στο διάγραμμα 3.6 τα αρχεία που τον απαρτίζουν είναι 4. Το **ClusteringAlgorithm** είναι ένα interface με μία συνάρτηση `cluster()`



Διάγραμμα 3.6: Ο φάκελος clustering

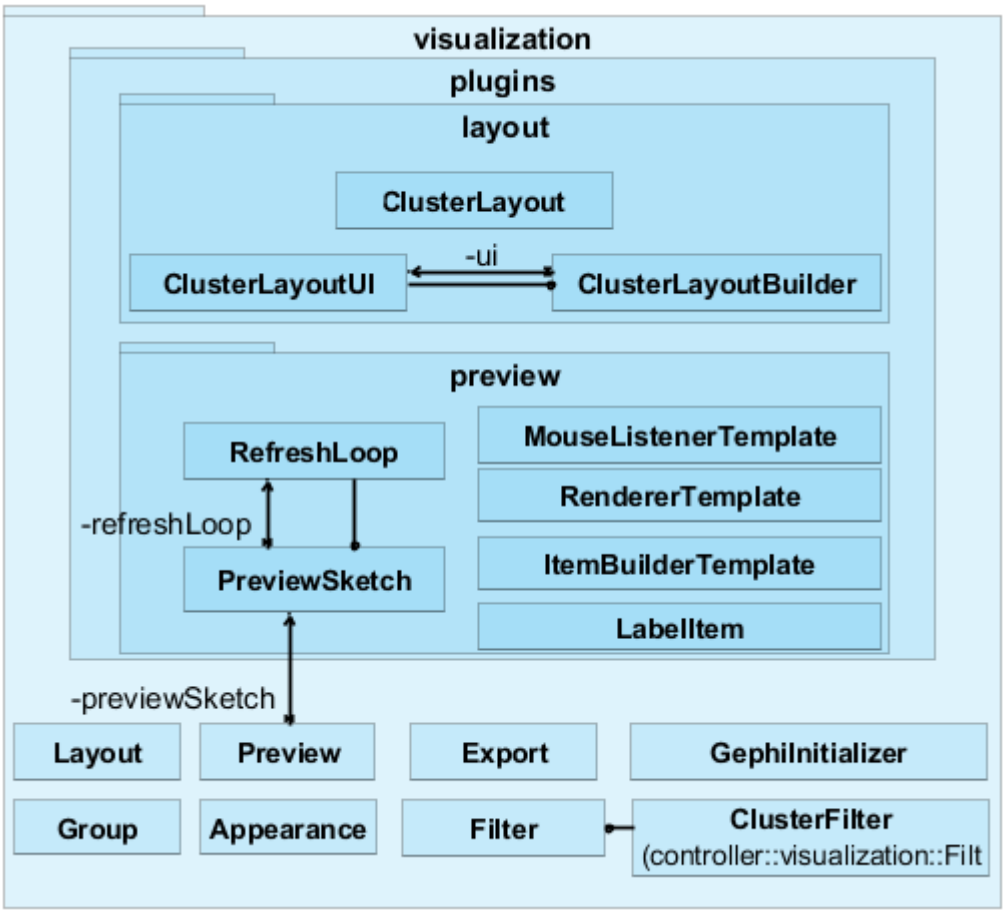
η οποία παίρνει σαν είσοδο το γράφο και το πλήθος τον επαναλήψεων, που έχει ορίσει ο χρήστης, για την εκτέλεση του αλγορίθμου.

Το **HillClimb** είναί μια υλοποίηση του **ClusteringAlgorithm**, για τον αλγόριθμο συσταδοποίησης αναρρίχησης λόφων. Το **NeightbourPartitionGenerator** είναι ένα εργαλείο που χρησιμοποιεί η **HillClimb** για τη δημιουργία γειτονικών τμημάτων και την τροφοδοτεί με τη βέλτιστη γειτονική διαμέριση του γράφου που βρέθηκε, ενώ το **RandomClusterGenerator** είναι ένα εργαλείο τυχαίας ταξινόμησης των κόμβων σε τυχαίο νούμερο υποσυστημάτων.

Στο διάγραμμα 3.7 εμφανίζεται η εσωτερική δομή του φακέλου visualization. Αυτός ο φάκελος περιέχει τα αρχεία που σχετίζονται με την οπτικοποίηση του γράφου. Στο αρχείο **GephiInitializer** δίνουμε ως είσοδο ένα γράφο ή ένα αρχείο και δημιουργεί ένα αντίστοιχο αντικείμενο γράφου της βιβλιοθήκης Gephi. Αυτό το αντικείμενο χρησιμοποιεί το αρχείο **Preview** για να δημιουργήσει ένα αντικείμενο **PreviewSketch**, το οποίο είναι ουσιαστικά η οπτικοποίηση του γράφου.

Το Preview λειτουργεί επίσης και ως controller για την επεξεργασία των ιδιοτήτων της προεπισκόπησης του γράφου (PreviewProperties). Τα αρχεία **Appearance**, **Layout**, **Filter** και **Group** είναι οι controller των υπηρεσιών εμφάνισης, διάταξης, φιλτραρίσματος και ομαδοποίησης αντίστοιχα, και χρησιμοποιούν τη βιβλιοθήκη της Gephi για την επεξεργασία και παραμετροποίηση του γράφου.

Το **ClusterFilter** είναι μία εσωτερική κλάση του controller **Filter** για την απλούστευση της διαδικασίας φιλτραρίσματος βάσει των ιδιοτήτων των υποσυστημάτων. Ενώ τέλος ο controller **Export** είναι υπεύθυνος για την εξατομίκευση του γράφου στον τύπο και θέση αρχείου που επέλεξε ο χρήστης.



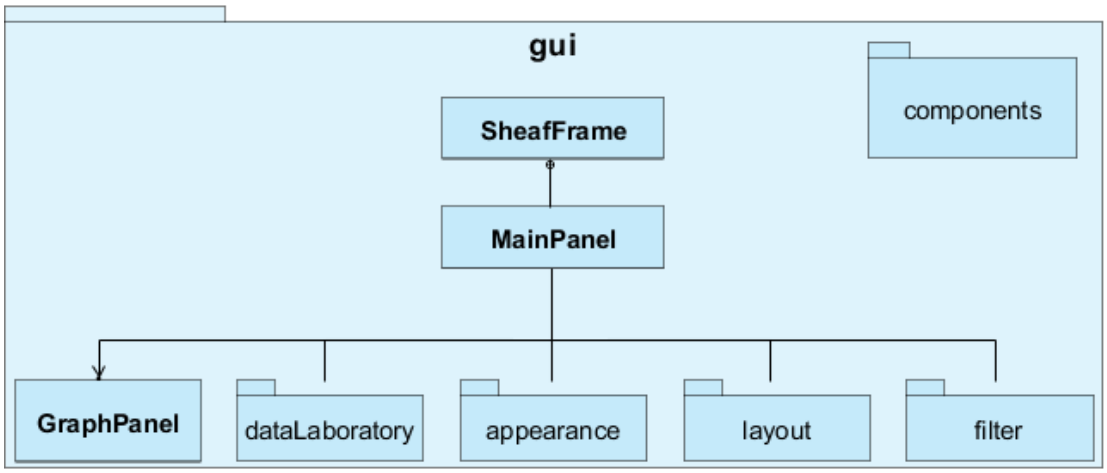
Διάγραμμα 3.7: Ο φάκελος visualization

Ο φάκελος visualization περιέχει επίσης το φάκελο plugin οποίος χωρίζεται με τη σειρά του στους **layout** και **preview**. Ο φάκελος **layout** περιέχει μια δικιά μας υλοποίηση της δομής *Layout* της Gephi και μία υλοποίηση της δομής *LayoutBuilder* της ίδιας βιβλιοθήκης καθώς και την εσωτερική της κλάση για τη διάταξη των κόμβων, βάσει του υποσυστήματος στο οποίο ανήκουν. Ο φάκελος **preview** περιέχει κλάσεις, τις οποίες πήραμε έτοιμες για την οπτικοποίηση και ανανέωση του γράφου. Επεξεργασθήκαμε μερικές από αυτές, για να προσθέσουμε επιπλέον λειτουργίες και δυνατότητες στο χρήστη σε ό,τι αφορά την προεπισκόπηση του γράφου.

3.2.3 Υποσύστημα GUI

Το υποσύστημα **GUI (graphic-user-interface)** είναι το αντίστοιχο του υποσυστήματος view του αρχιτεκτονικού μοντέλου MVC. Αποτελείται από το γραφικό περιβάλλον της διεπαφής χρήστη και είναι υπεύθυνο για τη λειτουργία αποστολής αιτημάτων στο υποσύστημα του controller για την επεξεργασία και παρουσίαση του model.

Για την υλοποίηση του view χρησιμοποιήσαμε τη βιβλιοθήκη **Swing**. Η Swing είναι ένα κομμάτι των *Java Foundation Classes (JFC)* της εταιρίας Oracle, ουσιαστικά είναι ένα API παροχής γραφικού περιβάλλοντος χρήστη (GUI) για προγράμματα Java. Το περιβάλλον της εφαρμογής μας αποτελείται από το συνδυασμό και την επέκταση των αντικειμένων αυτής της βιβλιοθήκης.



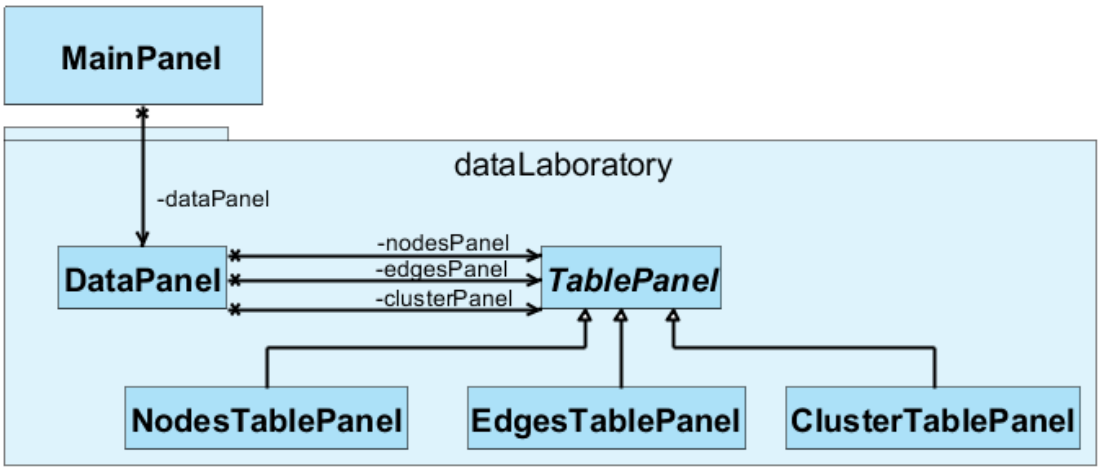
Διάγραμμα 3.8: Το υποσύστημα GUI

Στο διάγραμμα 3.8 παρουσιάζεται η γενικευμένη δομή του υποσυστήματος GUI. Παρόλο που το γραφικό περιβάλλον της εφαρμογής αποτελείται κυρίως από ένα παράθυρο (*JFrame*), έχουμε διαιρέσει τον πηγαίο κώδικα σε επιμέρους υποσυστήματα και αντικείμενα ανάλογα με τη χρήση τους, τόσο για την ευκολότερη κατανόηση του κώδικα όσο και για την επαναχρησιμοποίηση του.

Ενώ τα επιμέρους υποσυστήματα είναι ανεξάρτητα μεταξύ τους, παρατηρούμε ότι όλα συνδέονται με το αντικείμενο **MainPanel**. Μοναδική εξαίρεση των παραπάνω αποτελεί ο φάκελος **components** που περιλαμβάνει γραφικά στοιχεία της εφαρμογής όπως τα *CustomColorPicker*, *CustomFileChooser*, *DropDownButton* κτλ.

Αναλυτικότερα, το **SheafFrame** κληρονομεί το αντικείμενο *JFrame* του Swing και είναι το κύριο παράθυρο του εργαλείου Sheaf. Η **MainPanel** είναι εσωτερική κλάση του SheafFrame και κληρονομεί το αντικείμενο *JPanel*. Περιλαμβάνει όλα τα επιμέρους στοιχεία της εφαρμογής, εξ ου και η σύνδεση της με τα υπόλοιπα υποσυστήματα. Η **GraphPanel** κληρονομεί και αυτή το *JPanel* και αποτελείται από το πλαίσιο οπτικοποίησης του γράφου και δύο γραμμών εργαλείων. Οι controllers που της αντιστοιχούν είναι ο `Preview` και `Group`.

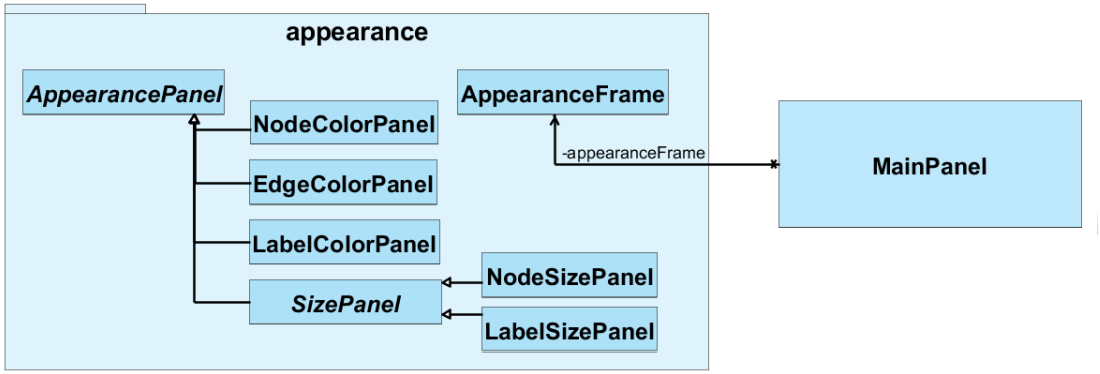
Το υποσύστημα **dataLaboratory** (Διάγραμμα 3.9) περιλαμβάνει ένα panel, το



Διάγραμμα 3.9: Ο φάκελος dataLaboratory

DataPanel, με πληροφορίες για τα στοιχεία του γράφου. Μέσα σε αυτό υπάρχουν τρεις όψεις, μία για τους κόμβους, μία για τις ακμές και τέλος μία για τα υποσυστήματα, οι οποίες περιλαμβάνονται στα **NodesTablePanel**, **EdgesTablePanel** και **ClusterTablePanel** αντίστοιχα.

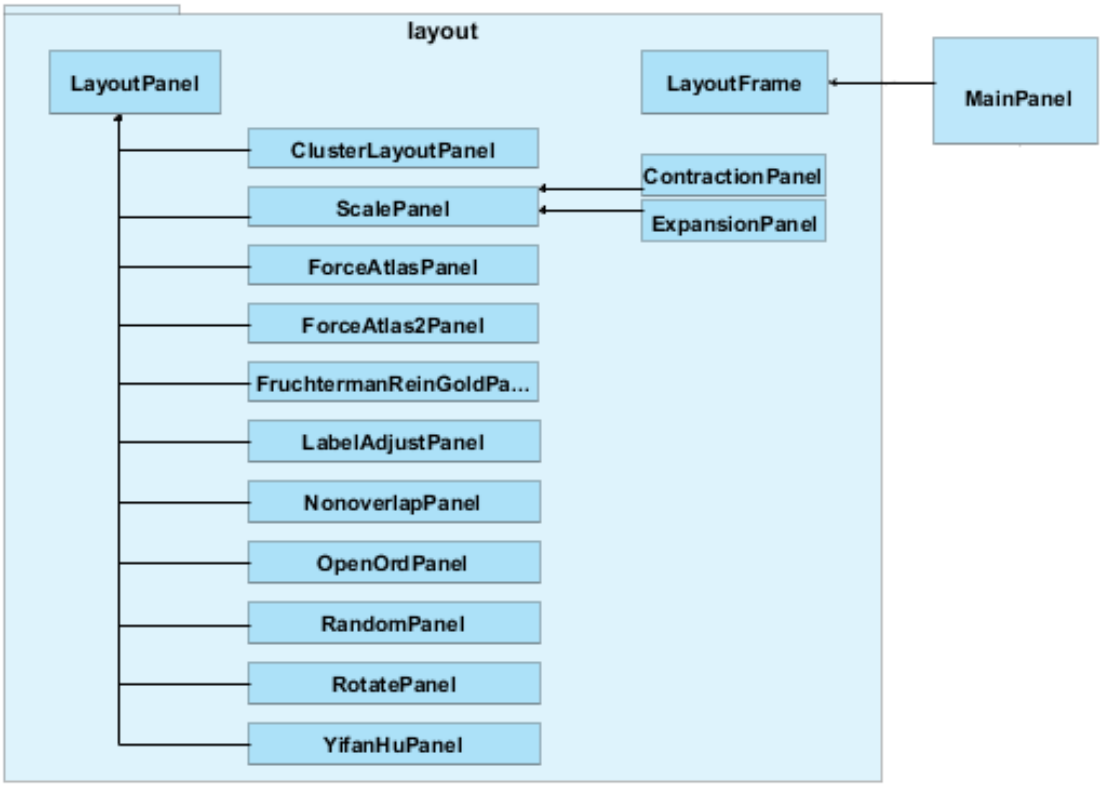
Λόγω των ομοιοτήτων τους προτιμήσαμε να συμπεριλάβουμε τα κοινά αυτών των panel σε μία αφηρημένη κλάση, την **TablePanel**, την οποία κληρονομούν, και να υλοποιήσουμε ξεχωριστά μόνο τις διαφορές τους. Το υποσύστημα του dataLaboratory επικοινωνεί με το model, ώστε να εμφανίσει τις ιδιότητες του μέσω του κεντρικού controller Project.



Διάγραμμα 3.10: Ο φάκελος appearance

Το υποσύστημα **appearance** (Διάγραμμα 3.10) αποτελείται από το **AppearanceFrame**, το οποίο κληρονομεί το αντικείμενο *JInternalFrame* και συμπεριλαμβάνει τα υπόλοιπα panel του υποσυστήματος. Λόγω της ομοιότητας μεταξύ τους και κάποιων λειτουργικών χαρακτηριστικών τα οποία μοιράζονται, δημιουργήσαμε μία αφηρημένη

κλάση την **AppearancePanel** και την κληρονομήσαμε από τις υπόλοιπες (**NodeColorPanel, EdgePanel, LabelColorPanel** και **SizePanel**), ώστε να επαναχρησιμοποιήσουμε το κοινό κώδικα. Με το ίδιο κριτήριο οι **NodeSizePanel** και **LabelSizePanel** κληρονομούν με τη σειρά τους την αφηρημένη κλάση **SizePanel**.



Διάγραμμα 3.11: Ο φάκελος layout

Στο διάγραμμα 3.11 παρουσιάζεται το υποσύστημα του **layout**. Σε αυτό ανήκει το **LayoutFrame**, το οποίο περιλαμβάνει δώδεκα panel με την περιγραφή και τις ιδιότητες ενός αλγόριθμου διάταξης το καθένα. Αυτά τα panel κληρονομούν τη κλάση **LayoutPanel**, η οποία αποτελείται από το σύνολο των στοιχείων και λειτουργιών που μοιράζονται μεταξύ τους τα panel που την κληρονομούν.

Τα αντικείμενα **ContractionPanel** και **ExpansionPanel** διαφέρουν μόνο στις αρχικές τιμές ιδιοτήτων τους και στις λειτουργίες τους, γι' αυτό κληρονομούν τη κλάση **ScalePanel**, που είναι μία επέκταση της LayoutPanel. Ο αντίστοιχος controller του υποσυστήματος είναι ο **Layout**.

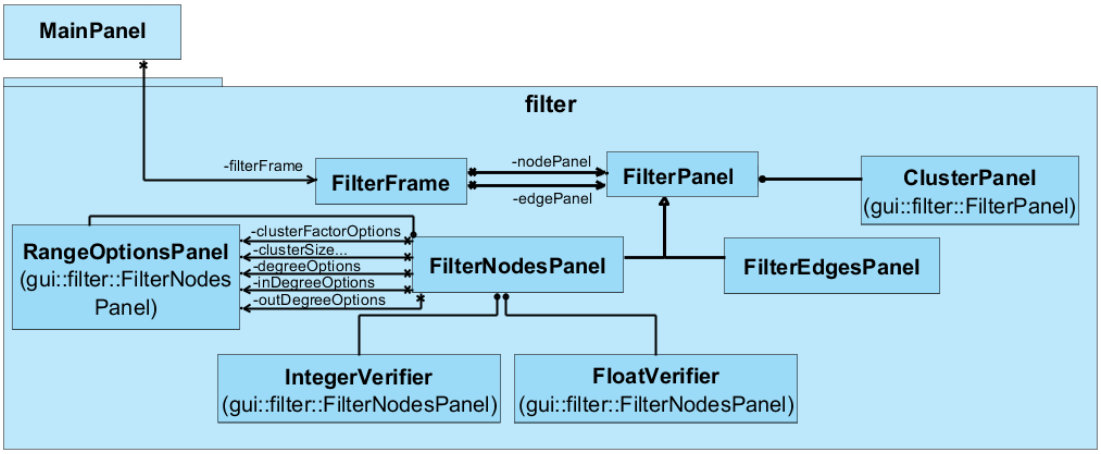
Όμοια με τα υποσυστήματα dataLaboratory και appearance, το υποσύστημα **filter** αποτελείται από το **FilterFrame**, το οποίο συμπεριλαμβάνει δύο επεκτάσεις της αφηρημένης κλάσης **FilterPanel**, τη **FilterNodesPanel** και **FilterEdgesPanel**, οι οποίες

περιέχουν τις επιλογές φιλτραρίσματος των κόμβων και ακμών αντίστοιχα.

Η **FilterPanel** έχει μία εσωτερική κλάση, τη **ClusterPanel**, η οποία είναι η υλοποίηση της αναπαράστασης των φίλτρων υποσυστημάτων του γράφου και χρησιμοποιείται από την **FilterNodesPanel** και **FilterEdgesPanel**.

Η **FilterNodesPanel** έχει και αυτή με τι σειρά της, τρεις εσωτερικές κλάσεις. Η **RangeOptionsPanel** χρησιμοποιείται για την επιλογή ενός πεδίου τιμών σε αρκετά φίλτρα, ενώ οι **IntegerVerifier** και **FloatVerifier** κληρονομούν τη κλάση *InputVerifier* και χρησιμοποιούνται για τον έλεγχο των πεδίων τιμών των φίλτρων για ακέραιους και δεκαδικούς αριθμούς αντίστοιχα.

Ο controller της είναι ο Filter και η δομή της εμφανίζεται στο διάγραμμα 3.12.



Διάγραμμα 3.12: Ο φάκελος filter

3.3 Επέκταση του Sheaf

Το εργαλείο Sheaf που αναπτύξαμε, υλοποιεί έναν προσαρμοσμένο αλγόριθμο αναρρίχησης λόφων για τη συσταδοποίηση γράφων, αξιολογώντας την ποιότητα των διαμερίσεων του βάσει του τύπου της TurboMQ που αναφέραμε στην ενότητα 2.2.4. Ο σχεδιασμός του όμως μας επιτρέπει τη μελλοντική του επέκταση και υλοποίηση επιπλέον αλγορίθμων με το ίδιο ή διαφορετικό κριτήριο αξιολόγησης της ποιότητας κατάτμησης. Σε αυτό το κεφάλαιο θα περιγράψουμε τη διαδικασία που πρέπει να ακολουθήσει κάποιος για την επέκταση του Sheaf.

Αρχικά ας θυμηθούμε ότι η TurboMQ ισούται με το σύνολο συντελεστών των επιμέρους υποσυστημάτων ($\sum$CF). Με γνώμονα αυτό, εισάγαμε στις συναρτήσεις προ-

σθαφαίρεσης στοιχείων της (`MDGImp` και `ClusterImpl`), τη λειτουργία αυτόματου υπολογισμού και ανανέωσης της MQ και CF αντίστοιχα.

Σε αυτό το κομμάτι συμβιβαστήκαμε στην απώλεια ευελιξίας μιας εναλλακτικής σχεδίασης, όπου η MQ θα ήταν ένα ξεχωριστό αντικείμενο και συνεπώς θα μπορούσαμε να αλλάξουμε ή να επιλέξουμε διαφορετικό τύπο για τον υπολογισμό της, έναντι μεγαλύτερης αποδοτικότητας, ταχύτητας υπολογισμού και ακρίβειας. Ας ξεκαθαρίσουμε επίσης ότι οι μοναδικές δομές που χρειάζεται για να λειτουργήσει τόσο η διαδικασία συσταδοποίησης όσο και εκείνη της εμφάνισης, είναι οι υλοποιήσεις των interface **MDG** και **Cluster**.

Έχοντας λοιπόν τα παραπάνω ως δεδομένα μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο που υλοποιήσαμε με διαφορετικές υλοποιήσεις των **MDG** και **Cluster**, για διαφορετικό τύπο της MQ. Αν η νέα μας MQ παραμένει το σύνολο από διαφορετικό τύπο CF, έχουμε μόνο να δημιουργήσουμε μία νέα υλοποίηση του **Cluster** που να περιλαμβάνει τους υπολογισμούς βάσει του καινούριου CF. Αν ο CF παραμένει ο ίδιος αλλά η MQ δεν ισούται με το σύνολο του, τότε δημιουργούμε μόνο την υλοποίηση του MDG.

Για την υλοποίηση περαιτέρω αλγορίθμων, είναι απαραίτητο η έξοδος του να είναι ένα αντικείμενο που να υλοποιεί το interface MDG, ενώ ο ίδιος ο αλγόριθμος να είναι υλοποίηση του interface **ClusteringAlgorithm**. Το **ClusteringAlgorithm** περιέχει μόνο μία συνάρτηση (την `cluster()`) που παίρνει ως είσοδο ένα αντικείμενο *Graph*, μία άδεια υλοποίηση *MDG* και μία *Cluster* για τον προσδιορισμό των δομών, ένα ακέραιο αριθμό *iterations* και δίνει ως έξοδο ένα αντικείμενο *MDG*. Το αντικείμενο **Graph** αποτελεί το σύνολο των στοιχείων του γράφου και είναι ανεξάρτητο της διαδικασίας συσταδοποίησης και της ποιότητας κατάτμησης, ενώ ο **iterations** αντιστοιχεί στον αριθμό επανάληψης του αλγορίθμου.

```
package controller.clustering;

interface ClusteringAlgorithm {
   MDG cluster(Graph graph, MDG mdg, Cluster cluster, int
      iterations);
}
```

Εφόσον προχωρήσουμε σε υλοποίηση περαιτέρω αλγορίθμων ή/και δομών που υποστηρίζουν τον υπολογισμό διαφορετικής MQ, θα χρειαστούν επίσης μερικές προσαρμογές στο υποσύστημα **GUI**, ώστε να μπορεί ο χρήστης να επιλέξει ανάμεσα τους.

Κεφάλαιο 4

Εγχειρίδιο χρήσης του εργαλείου Sheaf

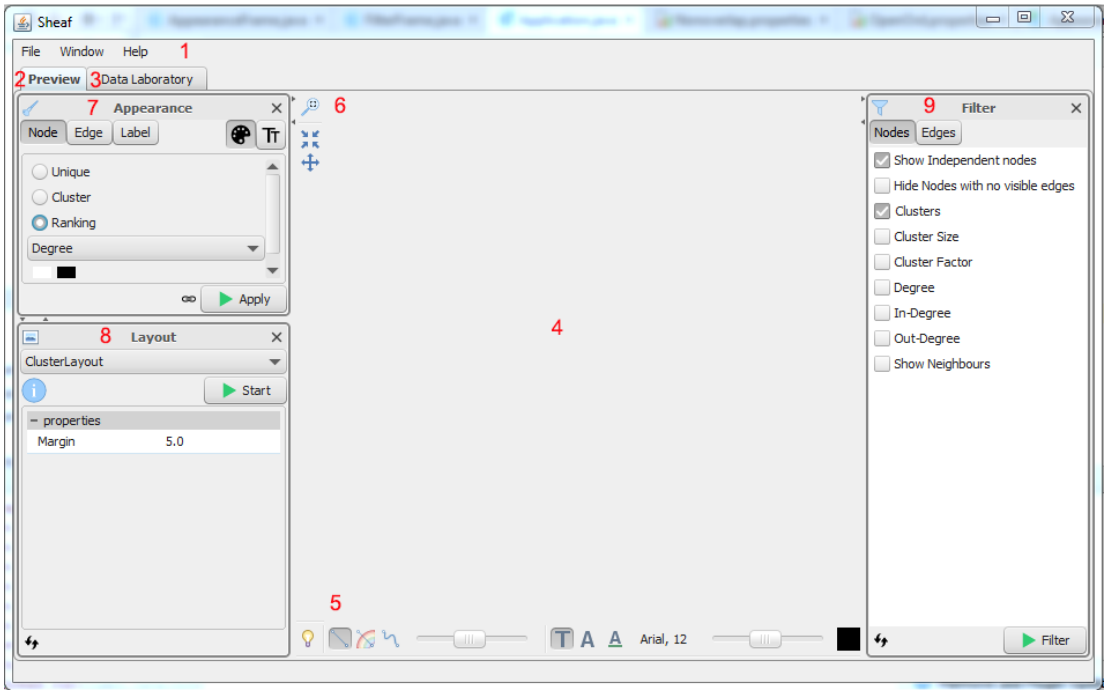
Σε αυτό το κεφάλαιο παρουσιάζεται το εργαλείο Sheaf στη μορφή που το αντιλαμβάνεται ο χρήστης και οδηγίες για την ορθή και πιο εύκολη χρήση του. Επίσης δίνεται ένα μικρό παράδειγμα χρήσης και περιγράφονται αναλυτικότερα οι δυνατότητες που δίνονται στο χρήστη για τις λειτουργίες συσταδοποίησης αλλά και για τις υπηρεσίες οπτικοποίησης και παραμετροποίησης τους.

Για την πιο εύκολη κατανόηση των προαναφερθέντων, παρέχουμε παρακάτω αριθμημένες εικόνες της διεπαφής χρήστη. Οι αριθμοί πάνω στις εικόνες αντιστοιχούν ένα κομμάτι της εικόνας με την περιγραφή που δίνεται παρακάτω και βοηθάει στην αποσαφήνιση τόσο του τρόπου όσο και του στόχου χρήσης του.

Στο διάγραμμα 4.1 παρουσιάζεται η διεπαφή χρήστη και τα επιμέρους τμήματα της.

1. Το κυρίως μενού της εφαρμογής, με τρεις βασικές επιλογές: File, Window και Help. Το υπο-μενού του File αναφέρεται αναλυτικότερα παρακάτω αλλά ουσιαστικά περιέχει τις επιλογές δημιουργίας καινούριου project, άνοιγμα υπάρχοντος αρχείου, αποθήκευσης και εξαγωγής του project. Το μενού Window δίνει την επιλογή εμφάνισης ή απόκρυψης των επιμέρους παραθύρων της εφαρμογής Appearance, Layout και Filter που εμφανίζονται στα νούμερα 7, 8, 9 του διαγράμματος 4.1 αντίστοιχα. Τέλος το μενού Help παρέχει κάποιες βασικές πληροφορίες για την εφαρμογή.

2. Η προεπισκόπηση (Preview) είναι μία από τις δύο βασικές καρτέλες της εφαρμογής και σε αυτή περιέχεται η οπτικοποιημένη μορφή του ομαδοποιημένου γρά-



Διάγραμμα 4.1: Διεπαφή χρήστη του εργαλείου Sheaf

φου και διάφορα εργαλεία για την παραμετροποίηση και διαμόρφωση αυτή της οπτικοποίησης.

3. Το εργαστήριο δεδομένων (Data Laboratory) περιέχει πληροφορίες για το γράφο και τα επιμέρους τμήματα του (κόμβους, ακμές, υποσυστήματα) με τη μορφή κειμένου.

4. Σε αυτό το σημείο βρίσκεται η περιοχή στην οποία εμφανίζεται η οπτικοποίηση του γράφου. Οι κόμβοι αναπαριστώνται με τη βοήθεια κυκλικών σχημάτων ενώ οι ακμές από γραμμές μεταξύ των κόμβων. Επίσης τον κάθε κόμβο συνοδεύει και η αντίστοιχη επιγραφή με το όνομα του κόμβου.

5. Μία από τις δύο γραμμές εργαλείων. Η συγκεκριμένη περιέχει εργαλεία για την επεξεργασία της εμφάνισης του γράφου, τα οποία αναφέρονται αναλυτικότερα σε παρακάτω ενότητα.

6. Η δεύτερη από τις γραμμές εργαλείων περιέχει τρεις επιλογές. Πρώτη είναι η επαναφορά του μεγέθους (reset zoom) στο γράφο και οι άλλες δύο επιτρέπουν στο χρήστη την επιλογή αναπαράστασης των υποσυστημάτων από ένα μόνο κόμβο για το κάθε υποσύστημα αντίστοιχα και το αντίστροφο.

7. Το παράθυρο της εμφάνισης (Appearance), δίνει στον χρήστη την δυνατότητα να επιλέξει ή να παραμετροποιήσει το χρώμα και το μέγεθος των στοιχείων του γράφου (κόμβοι, ακμές, επιγραφές).

8. Το παράθυρο διάταξης (Layout) επιτρέπει στο χρήστη να επιλέξει έναν από τους αλγόριθμους διάταξης και να τον εφαρμόσει στο γράφο με αποτέλεσμα οι κόμβοι του γράφου να αναδιαταχθούν στο χώρο ανάλογα με τον επιλεχθέντα αλγόριθμο.

9. Το παράθυρο φιλτραρίσματος αφήνει το χρήστη να διαλέξει και να εφαρμόσει τα δικά του κριτήρια στην εμφάνιση κόμβων και ακμών του γράφου.

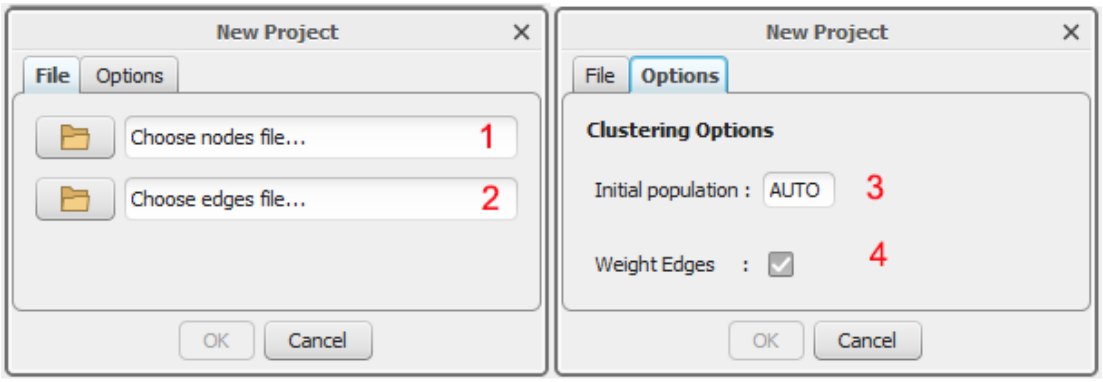
Στις ενότητες που ακολουθούν ασχολούμαστε με το κάθε τμήμα ξεχωριστά και αναλυτικότερα. Παραθέτουμε τις επιμέρους δυνατότητες του κάθε τμήματος, οδηγίες χρήσης τους και μικρά παραδείγματα χρήσης όπου θεωρούμε ότι χρειάζονται. Αξίζει να σημειωθεί ότι οι υπηρεσίες εμφάνισης, οπτικοποίησης, διάταξης και φιλτραρίσματος παρέχονται επί των πλείστων από το API της Gephi και οι παρακάτω οδηγίες αποτελούν μία συμπληρωματική μετάφραση των πληροφοριών που μας παρέχει εκείνη, με μερικές διευκρινήσεις και παραδείγματα από τη μεριά μας, που αποσκοπούν στη πιο εύκολη χρήση του εργαλείου που αναπτύξαμε.

4.1 Μικρό παράδειγμα χρήσης

Σε αυτήν την ενότητα θα περιγράψουμε τα βήματα για τη δημιουργία ενός νέου project, το άνοιγμα ενός ήδη υπάρχοντος, την αποθήκευση του και την εξαγωγή του σε μορφή εικόνας ή γράφου. Όλες οι παραπάνω διαδικασίες συμπεριλαμβάνονται στο μενού αρχείου (File).

Απαραίτητο κριτήριο για τη δημιουργία νέου project αποτελούν τα δύο CVS αρχεία με τις πληροφορίες των κόμβων και των ακμών όπως αναφέρονται στην ενότητα 1.2. Εφόσον έχουμε προμηθευτεί τα δύο αυτά αρχεία από τον συντακτικό αναλυτή παραμένει να τα εισάγουμε ως είσοδο στην εφαρμογή. Επιλέγουμε το μενού αρχείου (File) και μετέπειτα την επιλογή New project, παρατηρούμε ότι εμφανίζεται το παράθυρο του διαγράμματος 4.2. Σε αυτό το παράθυρο έχουμε τα εξής πεδία και συνεπώς βήματα:

1. Επιλέγουμε το αρχείο CVS κόμβων.



Διάγραμμα 4.2: Παράθυρο δημίουργίας νέου Project

2. Επιλέγουμε το αρχείο CVS ακμών.

3. Επιλέγουμε προαιρετικά τον αριθμό του αρχικού πληθυσμού που θα έχει ως είσοδο ο αλγόριθμος συσταδοποίησης. Αν δεν επιλέξουμε, η εφαρμογή επιλέγει για εμάς έναν προκαθορισμένο αριθμό ανάλογα με το μέγεθος του γράφου, όπου ο συμβιβασμός χρόνου εκτέλεσης και απόδοσης είναι ικανοποιητικός.

4. Επιλέγουμε αν το βάρος των ακμών θα συντελέσει στον υπολογισμό της ποιότητας κατάτμησης ή αν όχι. Η προεπιλογή της εφαρμογής είναι να λαμβάνει υπόψη της το βάρος, αλλά απλά απο-επιλέγοντάς την, αυτό αλλάζει και η εφαρμογή θεωρεί πλέον ότι όλες οι ακμές έχουν βάρος ίσο με ένα.

Εφόσον έχουν συμπληρωθεί τα υποχρεωτικά πεδία (1 και 2), ο χρήστης μπορεί να πατήσει OK και να ξεκινήσει η διαδικασία συσταδοποίησης. Στο κάτω δεξιό μέρος της εφαρμογής θα εμφανισθεί μία γραμμή προόδου που τον ενημερώνει για την ομαλή εξέλιξη της διαδικασίας. Στο πέρας αυτής της διαδικασίας η γραμμή προόδου εξαφανίζεται και συμπληρώνονται οι πίνακες στο Data laboratory με τα στοιχεία του γράφου όπως επίσης εμφανίζεται και ο γράφος στην καρτέλα της προεπισκόπησης. Πλέον ο χρήστης μπορεί να επεξεργασθεί το αποτέλεσμα, να το αποθηκεύσει για αργότερα ή να το εξωτερικεύσει με τη μορφή αρχείου της επιλογής του.

Όπως αναφέρθηκε στην ενότητα 1.2, ο χρήστης έχει την επιλογή να ανοίξει ένα αρχείο γράφου, είτε ήδη ομαδοποιημένου είτε όχι, αρκεί να είναι μίας των παρακάτω μορφών : gephi, gml, gdx, graphML, gefx ή vna. Επιλέγοντας *Άνοιγμα…(Open…)* από το μενού αρχείου ή πατώντας *Ctrl+O* και αργότερα επιλέγοντας το αρχείο. Αν το αρχείο περιέχει μη ομαδοποιημένο γράφο θα αρχίσει αυτόματα τη διαδικασία συσταδοποίη-

σης. Αν το αρχείο που θέλουμε να ανοίξουμε το έχουμε ξανανοίξει πρόσφατα μπορούμε να το ανοίξουμε απλά επιλέγοντας το από τις επιλογές του *Άνοιγμα πρόσφατου (Open recent)*.

Για την αποθήκευση του project χρησιμοποιείται η μορφή αρχείου Gephi, η οποία είναι και η μόνη ικανή για να διατηρήσει όλες τις αλλαγές που μπορεί να έχουμε επιφέρει στη μορφοποίηση του γράφου. Η αποθήκευση του project είναι μία εύκολη διαδικασία όπως και αυτή του ανοίγματος, απλά επιλέγουμε *Αποθήκευση (Save)* ή *Αποθήκευση ως (Save as)* τη θέση που θέλουμε να αποθηκεύσουμε το αρχείο και το όνομα του. Αν θέλουμε να αποθηκεύσουμε αλλαγές σε ήδη αποθηκευμένο αρχείο αρκεί να επιλέξουμε αποθήκευση ή να πατήσουμε *Ctrl+S*.

Η διαδικασία εξαγωγής του γράφου είναι ίδια με εκείνης της αποθήκευσης με μόνη διαφορά ότι έχουμε να επιλέξουμε αν θα τον εξωτερικεύσουμε ως αρχείο γράφου (gml, gdx, graphML, gefx, vna) ή εικόνας (pdf, png, svg) και αν θα εξωτερικεύσουμε όλο το γράφο ή μόνο τον ορατό.

4.2 Υπηρεσία οπτικοποίησης

Στο διάγραμμα 4.3 βλέπουμε την περιοχή του εργαλείου στην οποία εμφανίζεται η οπτικοποίηση του γράφου και τις δύο γραμμές εργαλείων που αναφέραμε παραπάνω. Σε αυτή την ενότητα θα ασχοληθούμε με τις δυνατότητες που μας παρέχουν αυτά τα εργαλεία, αλλά πρώτα θα αναφέρουμε κάποιες βασικές κινήσεις που μπορεί να κάνει ο χρήστης με το ποντίκι του πάνω στο γράφο.

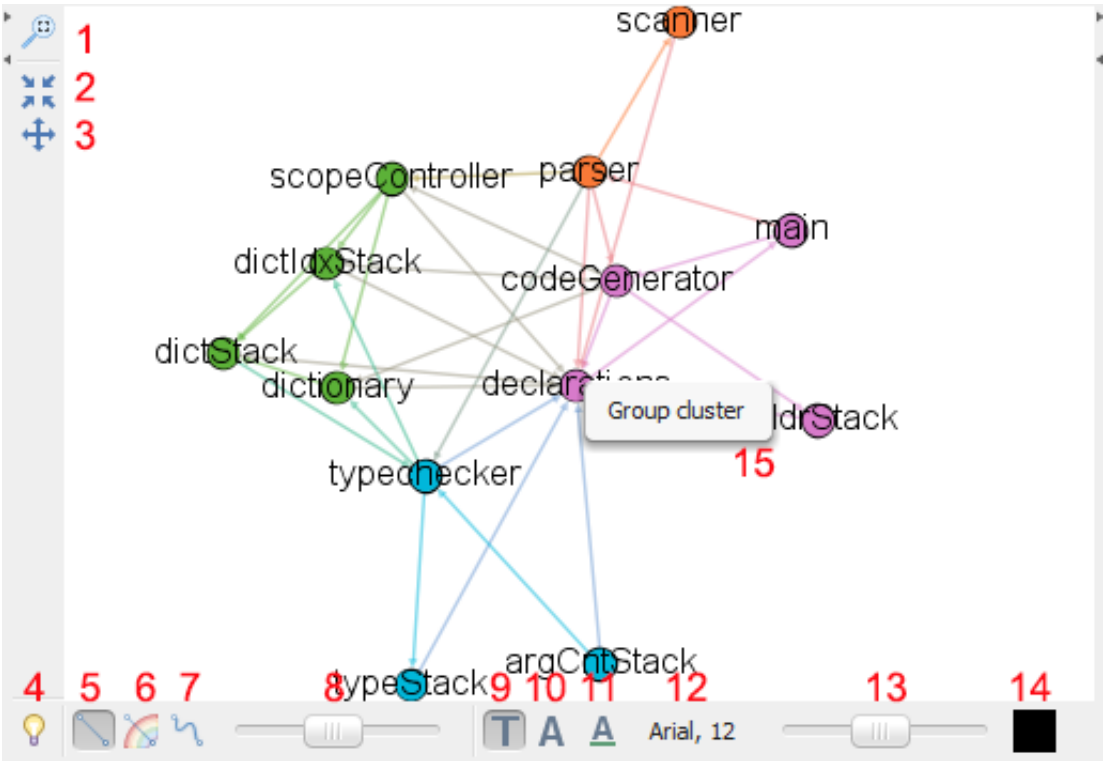
- **Scroll up/down** - Με αυτή τη κίνηση ο χρήστης ορίζει το ζουμ. Κάνοντας scroll up ο γράφος μεγεθύνεται στο χώρο ενώ αντίστοιχα κάνοντας scroll down σμικρύνεται.

- **Press and Drag σε κενό χώρο** - Πατώντας αριστερό ή δεξί κλικ στο κενό χώρο και σύροντας το ποντίκι ο χρήστης μπορεί να πλοηγηθεί στο χώρο.

- **Press and Drag σε κόμβο** - Με αυτό το συνδυασμό κινήσεων ο χρήστης μπορεί να μετακινήσει το επιλεγμένο κόμβο αλλάζοντας του τη θέση στο χώρο.

- **Αριστερό click σε κόμβο** - Εμφανίζει ένα παράθυρο με τα στοιχεία του επιλεγμένου κόμβου.

- **Δεξί click σε κόμβο** - Εμφανίζει ένα μενού για τον επιλεγμένο κόμβο.



Διάγραμμα 4.3: Οπτικοποίηση του γράφου

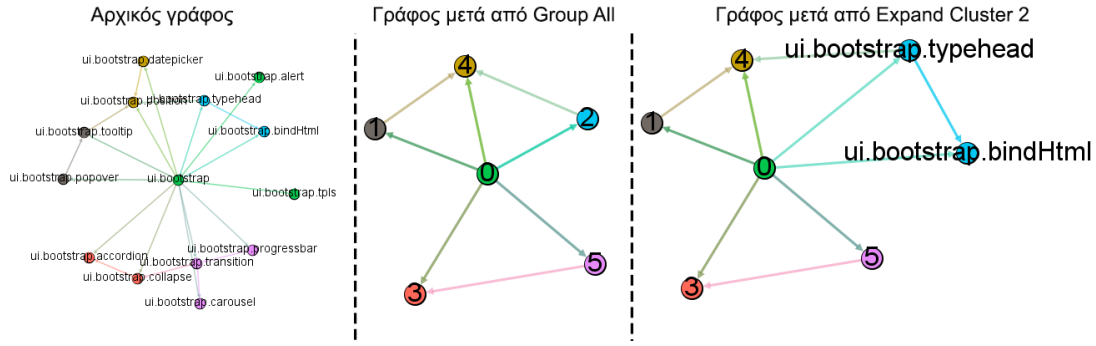
Ας εξετάσουμε τώρα τις επιλογές που μας δίνουν οι δύο γραμμές εργαλείων ξεκινώντας από την αριστερή κάθετη :

1. **Center on graph** - Επανακεντράρει το γράφο στο χώρο.
2. **Group all clusters** - Αντικαθιστά όλους του κόμβους ενός υποσυστήματος με ένα καινούριο κόμβο για όλα τα υποσυστήματα
3. **Expand all clusters** - Αντικαθιστά όλα τους κόμβους που αντιπροσωπεύουν υποσυστήματα με όλους τους αρχικούς κόμβους που απαρτίζουν το εν λόγω υποσύστημα.
4. **Background color** - Επιλέγει το χρώμα του φόντου. Με αριστερό κλικ εναλλάζει ανάμεσα σε λευκό και μαύρο ενώ με δεξί κλικ ανοίγει ένα παράθυρο επιλογής RGB χρώματος.
5. **Show edges** - Εμφανίζει ή αποκρύβει τις ακμές πάνω στο γράφο.

6. **Edge color mode** - Μας αφήνει να επιλέξουμε βάσει ποιου χαρακτηριστικού παίρνουν το χρώμα τους οι ακμές από τις παρακάτω επιλογές:
    - **Unique** - Χρησιμοποιεί ένα χρώμα για όλες τις ακμές.
    - **Source** - Χρησιμοποιεί το χρώμα του πηγαίου κόμβου τις ακμής.
    - **Target** - Χρησιμοποιεί το χρώμα του στόχου κόμβου τις ακμής.
    - **Mixed** - Χρησιμοποιεί μία ανάμειξη το χρώματος του πηγαίου και του στόχου κόμβου τις ακμής.
7. **Curve edges** - Επιλέγει αν οι ακμές θα είναι ευθείες γραμμές ή καμπύλες.
8. **Edge weight scale** - Ρυθμίζει το πλάτος των ακμών.
9. **Show labels** - Εμφανίζει ή αποκρύβει τις επιγραφές των κόμβων πάνω στο γράφο.
10. **Size mode** - Διαλέγει αν το μέγεθος των επιγραφών του κόμβου είναι ανάλογο με το μέγεθος του αντίστοιχου κόμβου:
    - **Fixed** - Το μέγεθος των επιγραφών είναι ανεξάρτητο του αντίστοιχου κόμβου
    - **Node size** - Το μέγεθος των επιγραφών είναι ανάλογο του αντίστοιχου κόμβου
11. **Label color mode** - Μας αφήνει να επιλέξουμε βάσει ποιού χαρακτηριστικού παίρνουν το χρώμα τους οι επιγραφές των κόμβων από τις παρακάτω επιλογές:
    - **Unique** - Χρησιμοποιεί το χρώμα που επιλέγεται από το παράθυρο εμφάνισης (Appearance).
    - **Object** - Χρησιμοποιεί το χρώμα του αντίστοιχου κόμβου
    - **Text** - Χρησιμοποιεί το χρώμα που επιλέγεται από το εργαλείο στο νούμερο 14 παρακάτω
12. **Font** - Επιλέγει τη γραμματοσειρά των επιγραφών των κόμβων.
13. **Font size scale** - Ρυθμίζει το μέγεθος των επιγραφών των κόμβων.
14. **Default color** - Επιλέγει το χρώμα των επιγραφών των κόμβων.

Στο νούμερο 15 του διαγράμματος βλέπουμε το μενού που εμφανίζεται αν κάνουμε δεξί κλικ σε ένα κόμβο και μας δίνει την επιλογή να αντικαταστήσουμε τους κόμβους του υποσυστήματος που ανήκει ο επιλεγμένος κόμβος με ένα μόνο κόμβο που αναπαριστά το υποσύστημα (group cluster).

Αυτό το μενού αντικαθίσταται σε περίπτωση που ο επιλεγμένος κόμβος αναπαριστά υποσύστημα και μας δίνει την επιλογή να επαναφέρουμε το υποσύστημα στην αρχική του μορφή (expand cluster) και να μετονομάσουμε (rename) το υποσύστημα. Η διαδικασία του group και expand cluster παρουσιάζεται καλύτερα στο διάγραμμα 4.4.



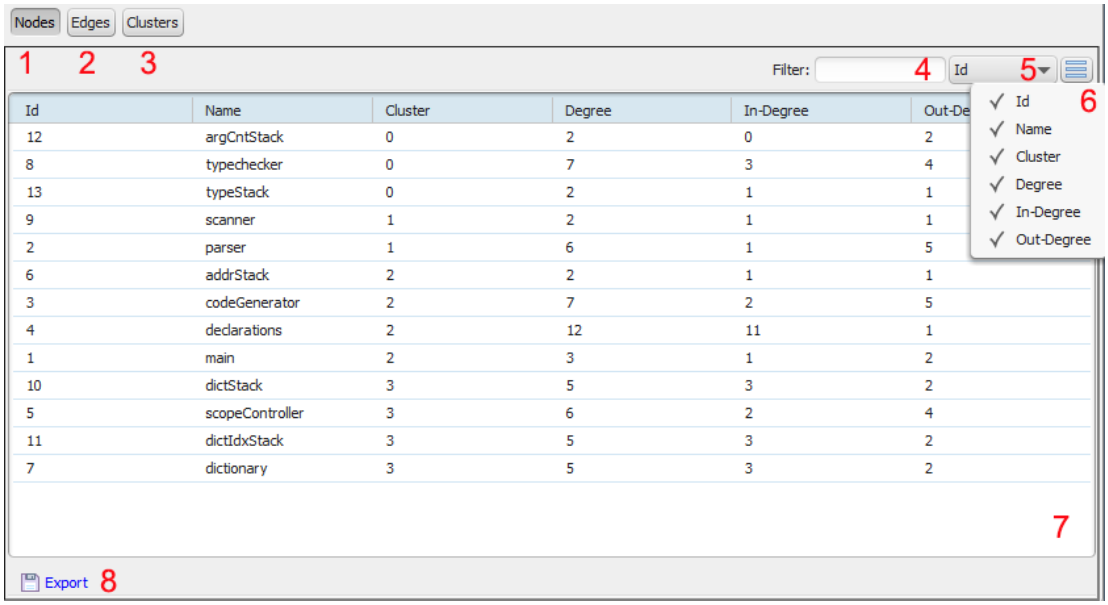
Διάγραμμα 4.4: Η διαδικασία του group και expand

4.3 Υπηρεσία δεδομένων

Σε αυτή την ενότητα βλέπουμε την υπηρεσία δεδομένων ή αλλιώς την καρτέλα του εργαστηρίου δεδομένων (Data Laboratory) που αναφέραμε στην αρχή αυτού το κεφαλαίου και αντιστοιχεί στο νούμερο 3 του διαγράμματος 4.1. Η υπηρεσία δεδομένων χωρίζεται σε τρεις βασικές κατηγορίες που την απαρτίζουν: τα δεδομένα κόμβων, ακμών και υποσυστημάτων.

Στο διάγραμμα 4.5 βλέπουμε την καρτέλα δεδομένων και έχουμε αριθμήσει τις περιοχές που την απαρτίζουν και παρακάτω σχολιάζουμε τη χρησιμότητα τους:

1. Εμφανίζει τον πίνακα δεδομένων κόμβων.
2. Εμφανίζει τον πίνακα δεδομένων ακμών.
3. Εμφανίζει τον πίνακα δεδομένων υποσυστημάτων.
4. Πεδίο εισαγωγής κειμένου για φιλτράρισμα των δεδομένων στους πίνακες.



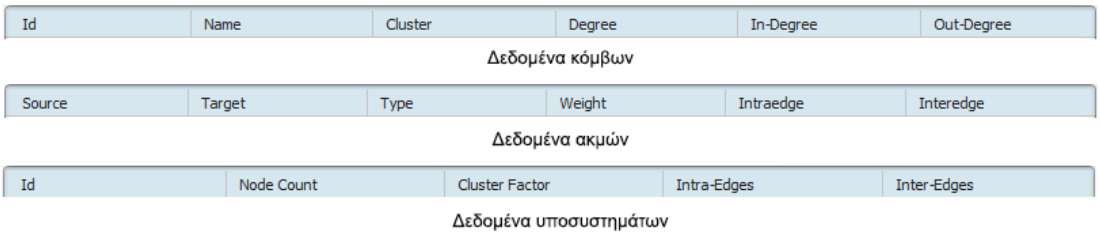
Διάγραμμα 4.5: Η υπηρεσία δεδομένων

5. Επιλογή στήλης για να εφαρμοσθεί το φιλτράρισμα κειμένου.

6. Επιλογή στηλών για εμφάνιση στο πίνακα δεδομένων.

7. Πίνακας δεδομένων.

8. Εξαγωγή του πίνακα δεδομένων σε CVS αρχείο.



Διάγραμμα 4.6: Διαθέσιμες στήλες

Στο διάγραμμα 4.6 βλέπουμε τις διαθέσιμες στήλες για το κάθε πίνακα δεδομένων. Αναλυτικότερα για το πίνακα δεδομένων των **κόμβων** (nodes) έχουμε τις εξής έξι στήλες:

- **Id** - Η αριθμητική ταυτότητα που χρησιμοποιεί η εφαρμογή για να αναγνωρίζει τους κόμβους.

- **Όνομα (Name)** - Το όνομα του κόμβου που δόθηκε ως είσοδος στην εφαρμογή.

- **Υποσύστημα (Cluster)** - Το όνομα του υποσυστήματος στο οποίο ταξινομήθηκε ο κόμβος.
- **Βαθμός (Degree)** - Το νούμερο των ακμών που έχουν ως πηγή ή τέρμα τον κόμβο.
- **Εσωτερικός βαθμός (In-Degree)** - Το νούμερο των ακμών που έχουν ως τέρμα τον κόμβο.
- **Εξωτερικός βαθμός (Out-Degree)** - Το νούμερο των ακμών που έχουν ως πηγή τον κόμβο.

Οι πληροφορίες που μας παρέχει ο αντίστοιχος πίνακας για τις **ακμές** (edges) είναι οι εξής:

- **Πηγαίος κόμβος (Source)** - το id του κόμβου από τον οποίο ξεκινάει η ακμή.
- **Στόχος κόμβος (Target)** - το id του κόμβου στον οποίο τερματίζει η ακμή.
- **Τύπος ακμής (Type)** - ο τύπος της ακμής (κατευθυνόμενη ακμή ή μη).
- **Βάρος ακμής (Weight)** - ο αριθμός που αντιστοιχεί στο βάρος της ακμής.
- **Intraedge** - αν η ακμή έχει πηγαίο και στόχο κόμβο που ανήκουν στο ίδιο υποσύστημα η τιμή του είναι αληθής (true), διαφορετικά είναι ψευδής (false).
- **Interedge** - αν η ακμή έχει πηγαίο και στόχο κόμβο που ανήκουν σε διαφορετικά υποσυστήματα η τιμή του είναι αληθής (true), διαφορετικά είναι ψευδής (false).

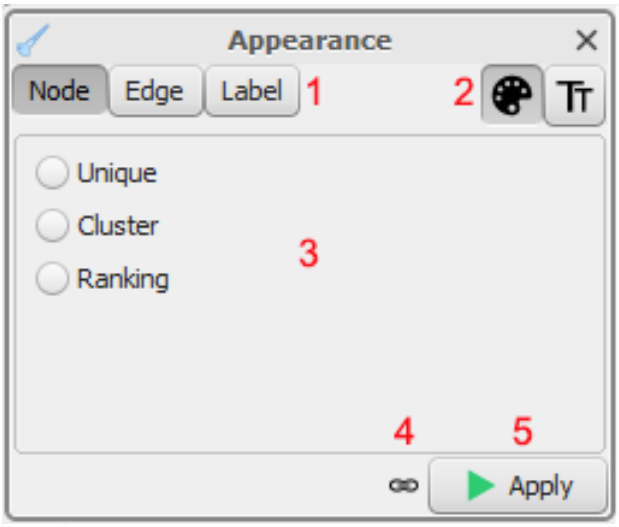
Τέλος, έχουμε τα δεδομένα για τα δημιουργηθέντα **υποσυστήματα** (clusters) που παρουσιάζονται στις παρακάτω στήλες:

- **Id** - Η αλφαριθμητική ταυτότητα του υποσυστήματος (η εφαρμογή αρχικά δίνει μία αριθμητική τιμή αλλά ο χρήστης έχει τη δυνατότητα αλλαγής της).
- **Αριθμός κόμβων (Node Count)** - Ο αριθμός των κόμβων που συντελούν το σύνολο του υποσυστήματος.

- **Συντελεστής υποσυστήματος (Cluster Factor)** - Η κανονικοποιημένη αναλογία μεταξύ του συνολικού βάρους των εσωτερικών ακμών (ακμών εντός του υποσυστήματος) και το ήμισυ του συνολικού βάρους των εξωτερικών ακμών (ακμές που εισέρχονται ή εξέρχονται στο υποσύστημα)
- **Intra-Edges** - Ο αριθμός των ακμών που έχουν τόσο πηγαίο όσο και στόχο κόμβο, που ανήκουν στο εν λόγω υποσύστημα.
- **Inter-Edges** - Ο αριθμός των ακμών που έχουν πηγαίο ή στόχο κόμβο, ο οποίος ανήκει στο εν λόγω υποσύστημα.

4.4 Υπηρεσία εμφάνισης

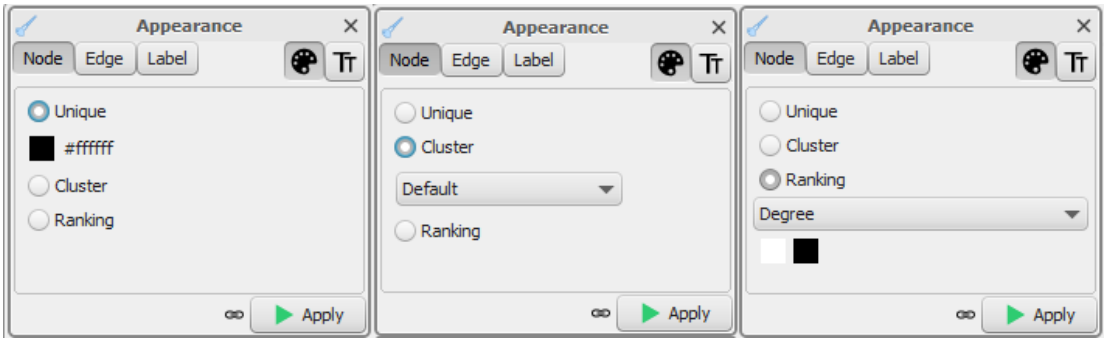
Το παράθυρο εμφάνισης (Appearance) μας δίνει τη δυνατότητα να επεξεργασθούμε τα δύο βασικά χαρακτηριστικά των συστατικών οπτικοποίησης του γράφου, το χρώμα και το μέγεθος. Μπορούμε να δώσουμε μία τιμή για όλο το πλήθος ενός συστατικού είτε να παραμετροποιήσουμε τις τιμές που θα πάρουν μέσα σε ένα σύνολο τιμών που ορίζουμε ανάλογα με τα χαρακτηριστικά τους. Ενώ για τους κόμβους και τις επιγραφές έχουμε την επιλογή να ορίσουμε το χρώμα και το μέγεθος, για τις ακμές μπορούμε μόνο το χρώμα. Το πλάτος των ακμών είναι ανάλογο του βάρους τους, η μόνη διαμόρφωση που μπορούμε να του επιβάλουμε είναι από την δεύτερη γραμμή εργαλείων της υπηρεσίας οπτικοποίησης που προαναφέρθηκε.



Διάγραμμα 4.7: Παράθυρο εμφάνισης

Στο διάγραμμα 4.7 βλέπουμε τις βασικές περιοχές του παράθυρου εμφάνισης. Το σημείο 1 μας αφήνει να επιλέξουμε πιο συστατικό θέλουμε να επεξεργασθούμε από τα παρακάτω: κόμβους (node), ακμές(edge) και επιγραφές (label), ενώ από το σημείο 2 διαλέγουμε την εμφάνιση επιλογών για την επεξεργασία χρώματος ή μεγέθους. Η περιοχή που αριθμείται με το νούμερο 3 είναι η περιοχή εμφάνισης των εκάστοτε επιλογών, οι διάφορες επιλογές της κάθε καρτέλας περιγράφονται αναλυτικότερα παρακάτω και η δυνατότητα εισαγωγής τιμής για τη κάθε μία εμφανίζεται μετά την επιλογή του αντίστοιχου κουμπιού. Στο νούμερο 5 έχουμε το κουμπί εφαρμογής (Apply) το οποίο πατάμε αφού έχουμε επιλέξει τις τιμές για να εφαρμοσθούν. Εναλλακτικά μπορούμε να επιλέγουμε το κουμπί αυτόματου μετασχηματισμού στο σημείο 4 πριν επιλέξουμε τιμές για άμεση εφαρμογή των επιλογών μας.

4.4.1 Επιλογή χρώματος κόμβων



Διάγραμμα 4.8: Επιλογές χρώματος κόμβων

Στο διάγραμμα 4.8 βλέπουμε τις τρεις επιλογές που μας δίνονται για την επεξεργασία του χρώματος των κόμβων:

- **Μοναδικό χρώμα (Unique)**

  Το χρώμα που επιλέγουμε εφαρμόζεται σε όλους τους κόμβους του γράφου.

- **Χρώμα υποσυστήματος (Cluster)**

  Όλοι οι κόμβοι του ίδιου υποσυστήματος έχουν ίδιο χρώμα μεταξύ τους και διαφορετικό από όλα τα υπόλοιπα υποσυστήματα. Διαλέγουμε μία παλέτα από τις παρακάτω διαθέσιμες του διαγράμματος 4.9 και η εφαρμογή διαλέγει τόσα χρώματα όσα και τα υποσυστήματα και τα αντιστοιχεί στο καθένα μοναδικά.

Διάγραμμα 4.9: Διαθέσιμες παλέτες

- **Χρώμα κατάταξης (Ranking)**

  Οι κόμβοι παίρνουν χρώμα ανάλογα με το βαθμό επιλογής μας. Το χρώμα τους μπορεί να είναι ένα από τα δύο που επιλέξαμε ή ένα ανάμεσα σε εκείνα ανάλογα με το βαθμό τους. Μπορούμε να επιλέξουμε ανάμεσα στους:

  – **Συνολικό βαθμό (Degree)** - το σύνολο όλων των ακμών που έχουν άκρο τους το κόμβο.

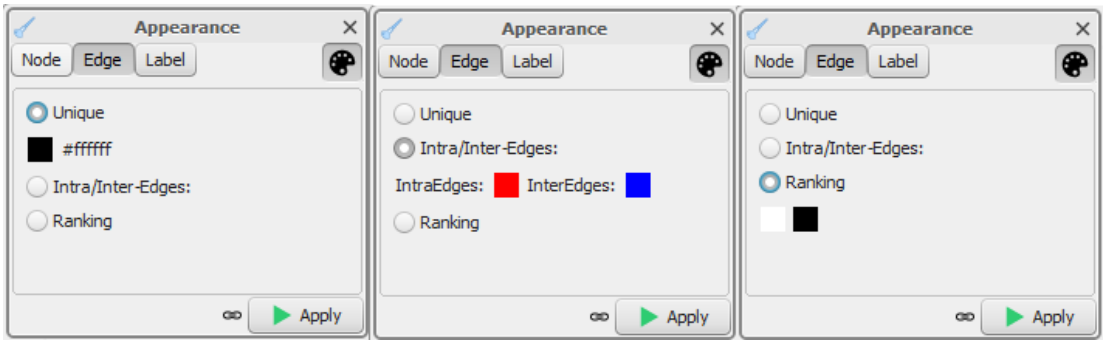
  – **Βαθμό εισόδου (In-Degree)** - το σύνολο των ακμών που τερματίζουν στο κόμβο.

  – **Βαθμό εξόδου (Out-Degree)** - το σύνολο των ακμών που ξεκινούν από τον κόμβο.

  Το αριστερό χρώμα επιλογής μας αντιστοιχεί στη μικρότερη τιμή του επιλεχθέντος βαθμού ενώ το δεξί στη μέγιστη.

4.4.2 Επιλογή χρώματος ακμών

Όπως βλέπουμε από το διάγραμμα 4.10 έχουμε και για τις ακμές τρεις επιλογές χρώματος:



Διάγραμμα 4.10: Επιλογές χρώματος ακμών

- **Μοναδικό χρώμα (Unique)**

  Το χρώμα που επιλέγουμε εφαρμόζεται σε όλες τις ακμές του γράφου.

- **Intra/Inter-Edges**

  Επιλέγουμε ένα χρώμα για τις intraedges και ένα για τις interedges. Το χρώμα που επιλέξαμε για τις intraedges εφαρμόζεται σε όλες τις ακμές ανάμεσα σε κόμβους του ίδιου υποσυστήματος ενώ στις υπόλοιπες το χρώμα που επιλέξαμε για τις interedges.

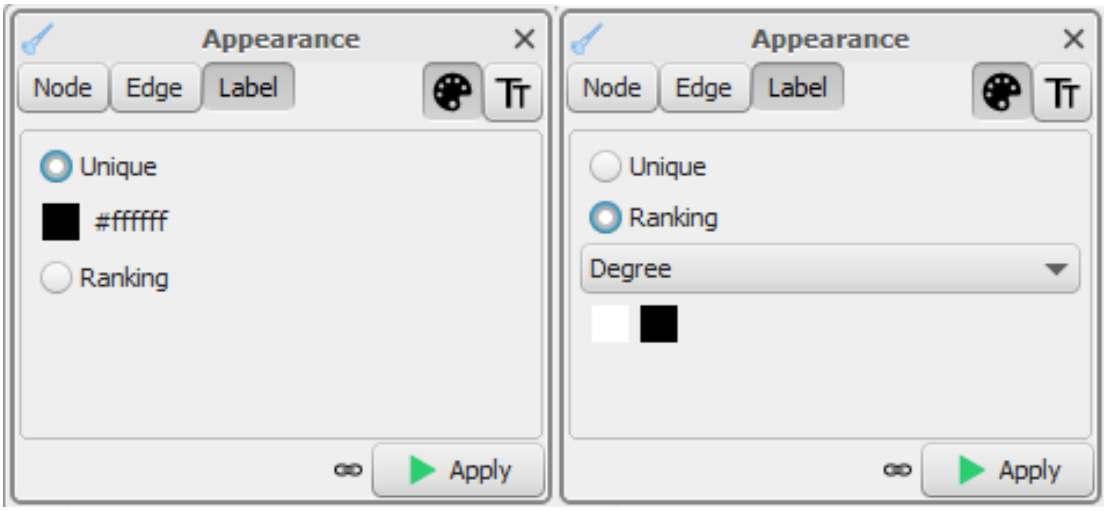
- **Χρώμα κατάταξης (Ranking)**

  Το χρώμα των ακμών υπολογίζεται ανάμεσα στα δύο επιλεχθέντα χρώματα ανάλογα με το βάρος τους. Το αριστερό χρώμα επιλογής μας αντιστοιχεί στη μικρότερη τιμή του βάρους ενώ το δεξί στη μέγιστη.

Αξίζει να επαναληφθεί ότι το χρώμα των ακμών μπορούμε να το αλλάζουμε και βάσει των κόμβων των οποίων ενώνει, από την γραμμή εργαλείων της υπηρεσίας οπτικοποίησης χρησιμοποιώντας το εργαλείο *edge color mode*.

4.4.3 Επιλογή χρώματος επιγραφών

Οι επιγραφές (labels) δεν είναι από τα βασικά συστατικά του γράφου (κόμβοι και ακμές). Ο στόχος τους είναι να ξεχωρίζουμε τους κόμβους μεταξύ τους. Συνεπώς οι



Διάγραμμα 4.11: Επιλογές χρώματος επιγραφών

επιγραφές δεν έχουν δικά τους χαρακτηριστικά για να παραμετροποιήσουμε την εμφάνιση τους, γι' αυτό και χρησιμοποιούμε τα χαρακτηριστικά των κόμβων στους οποίους αντιστοιχούν. Σε αντίθεση με τους κόμβους και τις ακμές, οι επιγραφές έχουν μόνο δύο επιλογές χρώματος όπως βλέπουμε και στο διάγραμμα 4.11.

- **Μοναδικό χρώμα (Unique)**

  Το χρώμα που επιλέγουμε εφαρμόζεται σε όλες τις επιγραφές του γράφου.

- **Χρώμα κατάταξης (Ranking)**

  Οι επιγραφές παίρνουν χρώμα ανάλογα με το βαθμό των κόμβων στον οποίο αντιστοιχούν, που επιλέγουμε. Το χρώμα τους μπορεί να είναι ένα από τα δύο που επιλέξαμε ή ένα ανάμεσα σε εκείνα ανάλογα με το βαθμό τους. Μπορούμε να επιλέξουμε ανάμεσα στους:

  – **Συνολικό βαθμό (Degree)** - το σύνολο όλων των ακμών που έχουν άκρο τους τον κόμβο της επιγραφής.

  – **Βαθμό εισόδου (In-Degree)** - το σύνολο των ακμών που τερματίζουν στον κόμβο της επιγραφής.

  – **Βαθμό εξόδου (Out-Degree)** - το σύνολο των ακμών που ξεκινούν από τον κόμβο της επιγραφής.

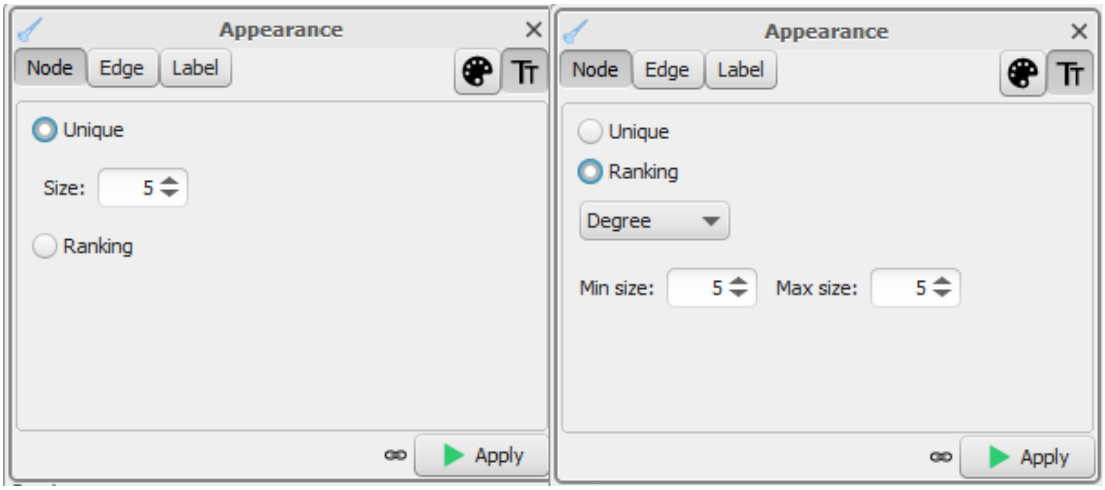
  Το αριστερό χρώμα επιλογής μας, αντιστοιχεί στη μικρότερη τιμή του επιλεχθέντος βαθμού ενώ το δεξί στη μέγιστη.

Αν θέλουμε οι επιγραφές να έχουν ίδιο χρώμα με τον κόμβο στον οποίο αντιστοιχούν δεν έχουμε παρά μόνο να επιλέξουμε την επιλογή Object από το εργαλείο *label color mode* της υπηρεσίας οπτικοποίησης, ενώ αν θέλουμε να επικρατήσει το *default color* της γραμμής εργαλείων διαλέγουμε την επιλογή Text.

4.4.4 Επιλογή μεγέθους

Όπως προαναφέραμε η επιλογή μεγέθους είναι διαθέσιμη μόνο για τους κόμβους και τις επιγραφές τους. Επίσης αναφέραμε ότι οι επιγραφές μοιράζονται τα χαρακτηριστικά των κόμβων στους οποίους αντιστοιχούν λόγω έλλειψης δικών τους.

Για τους λόγους αυτούς οι επιλογές επεξεργασίας του μεγέθους και των δύο είναι ίδιες. Παρακάτω εξηγούμε τις δύο αυτές κοινές επιλογές, μόνο για τους κόμβους εφόσον δεν αλλάζουν σε κάτι με τις αντίστοιχες για τις επιγραφές.



Διάγραμμα 4.12: Επιλογές μεγέθους

- **Μοναδικό μέγεθος (Unique)**

  Το μέγεθος που επιλέγουμε εφαρμόζεται σε όλους τους κόμβους του γράφου.

- **Μέγεθος κατάταξης (Ranking)**

  Το μέγεθος των κόμβων υπολογίζεται ανάλογα με το βαθμό επιλογής μας και ανάμεσα στο πεδίο ορισμού που επιλέγουμε. Μπορούμε να επιλέξουμε ανάμεσα στους εξής βαθμούς:

  – **Συνολικό βαθμό (Degree)** - το σύνολο όλων των ακμών που έχουν άκρο τους το κόμβο.

- **Βαθμό εισόδου (In-Degree)** - το σύνολο των ακμών που τερματίζουν στο κόμβο.
- **Βαθμό εξόδου (Out-Degree)** - το σύνολο των ακμών που ξεκινούν από τον κόμβο.

Το αριστερό πεδίο αντιστοιχεί στη μικρότερη τιμή του επιλεχθέντος βαθμού ενώ το δεξί στη μέγιστη.

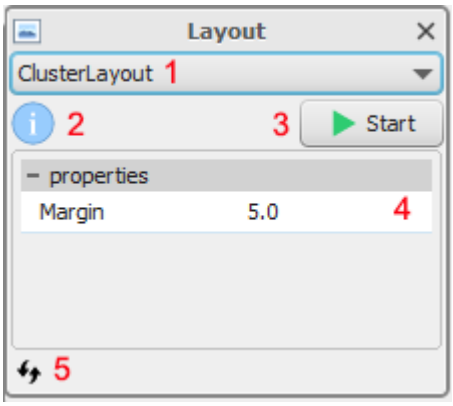
Να θυμίσουμε ότι το μέγεθος των επιγραφών εξαρτάται και από άλλους παράγοντες, όπως το *label size mode*, *font size scale* και το font size που επιλέγουμε από το εργαλείο *font* της υπηρεσίας οπτικοποίησης. Αν θέλουμε να ξαναεφαρμοσθούν οι αλλαγές που κάναμε από το παράθυρο εμφάνισης διαλέγουμε την επιλογή Fixed του label size mode. Αν θέλουμε να αλλάξουμε το μέγεθος των κόμβων και των επιγραφών βάσει του βαθμού τους με τιμές στο ίδιο πεδίο ορισμού είναι προτιμότερο απλά να χρησιμοποιήσουμε το παράθυρο εμφάνισης για τους κόμβους και την επιλογή Node size του label size mode για τις επιγραφές. Το font size scale και το font size που επιλέγουμε από το εργαλείο font απλά συνδράμουν στην αυξομείωση του μεγέθους γραμματοσειράς σε όλο το σύνολο των επιγραφών.

4.5 Υπηρεσία διάταξης

Η υπηρεσία διάταξης μας επιτρέπει να επανατοποθετήσουμε τους κόμβους στο χώρο με αυτοματοποιημένο τρόπο σύμφωνα με έναν αλγόριθμο επιλογής μας. Επίσης μπορούμε να παραμετροποιήσουμε τις ιδιότητες του εκάστοτε αλγορίθμου για να προσαρμόσουμε το αποτέλεσμα στις ανάγκες του γράφου μας ή για να δημιουργήσουμε μία πιο κατανοητή όψη του.

Ο χρήστης μπορεί να υποβάλει το γράφο στη διαδικασία διάταξης από το παράθυρο διάταξης (Layout), το οποίο και βλέπουμε στο διάγραμμα 4.13. Αναλυτικότερα για το παράθυρο διάταξης σύμφωνα με την αρίθμηση πάνω στο διάγραμμα:

1. **Επιλογή αλγόριθμου διάταξης** - Σχολιάζουμε τους διαθέσιμους αλγόριθμους και ιδιότητες τους ξεχωριστά παρακάτω.
2. **Πληροφορίες σχετικά με τον αλγόριθμο** - Εμφανίζονται αφήνοντας το ποντίκι να αιωρηθεί από πάνω.



Διάγραμμα 4.13: Παράθυρο διάταξης

3. **Κουμπί εκκίνησης του αλγορίθμου** - Μερικοί αλγόριθμοι τερματίζουν στο πρώτο βήμα ενώ κάποιοι άλλοι μετά από έναν αριθμό επαναλήψεων, ενώ υπάρχουν και εκείνοι που συνεχίζουν μέχρι να τους διακόψει ο χρήστης. Για τους δύο τελευταίους εφόσον έχουμε πατήσει το κουμπί εκκίνησης παρατηρούμε ότι αυτό δίνει τη θέση του σε ένα κουμπί τερματισμού. Ο χρήστης μπορεί να τερματίσει τον αλγόριθμο διάταξης οποιαδήποτε στιγμή θέλει. Η κάθε επανάληψη του αλγορίθμου εφαρμόζεται άμεσα και είναι ορατή στο γράφο που μας παρέχει η υπηρεσία οπτικοποίησης.

4. **Περιοχή ιδιοτήτων** - Σε αυτή τη περιοχή εμφανίζονται οι ιδιότητες του κάθε αλγορίθμου που μπορούμε να παραμετροποιήσουμε με την προκαθορισμένη τιμή τους. Αριστερά έχουμε το όνομα της ιδιότητας, αφήνοντας το ποντίκι πάνω εμφανίζονται και λεπτομέρειες για τη κάθε μία, και δεξιά η τιμή της που μπορούμε να επεξεργασθούμε.

5. **Κουμπί επαναφοράς** - Επαναφέρει τις τιμές όλων των ιδιοτήτων στη προκαθορισμένη τιμή τους.

4.5.1 Αλγόριθμοι διάταξης

Ας εξετάσουμε τώρα το κάθε αλγόριθμο διάταξης ξεχωριστά και τις ιδιότητες του:

- **Διάταξη υποσυστήματος (Cluster Layout)**

  Συγκεντρώνει τους κόμβους του ιδίου υποσυστήματος γύρω από τον κόμβο με τον μεγαλύτερο αριθμό ακμών. Έχει μόνο μία ιδιότητα:

  - **Περιθώριο (Margin)** - Η απόσταση ανάμεσα στους κόμβους του ίδιου υποσυστήματος. Μία μεγάλη τιμή δημιουργεί ένα πιο σποραδικό γράφημα και δε συνιστάται.

- **Συστολή (Contraction)** Ο αλγόριθμος συστολής, όπως είναι φανερό και από το όνομα του συστέλλει τη διάταξη των κόμβων του γράφου γύρο από το κέντρο της. Έχει μία παραμετροποιήσιμη ιδιότητα, τον:

  - **Συντελεστή κλίμακας (scale factor)** - Για την επιτυχή συστολή του γράφου πρέπει να πάρει τιμή στο πεδίο ορισμού (0,1) χωρίς τις τιμές των άκρων του πεδίου. Σε περίπτωση που χρησιμοποιήσουμε το μηδέν ως τιμή όλοι οι κόμβοι τοποθετούνται στο κέντρο και επικαλύπτονται, ενώ χρησιμοποιώντας το ένα το αποτέλεσμα παραμένει το ίδιο.

- **Διαστολή (Expansion)**

  Ο αλγόριθμος διαστολής, σε αντίθεση με τον αλγόριθμο συστολής, διαστείλει τη διάταξη των κόμβων του γράφου γύρο από το κέντρο της. Έχει μία παραμετροποιήσιμη ιδιότητα τον:

  - **Συντελεστή κλίμακας (scale factor)** - Για την επιτυχή διαστολή του γράφου πρέπει να πάρει τιμή μεγαλύτερη του ενός.

- **ForceAtlas** Η ForceAtlas κάνει τους γράφους πιο συμπαγείς, ευανάγνωστους και ικανούς να δείξουν τις αρχές πιο κεντρικές από τους κόμβους (επιλογή Distrib. Attraction). Η αυτόματη σταθεροποίηση βελτιώνει τη σύγκλιση στο τέλος της διάταξης. Οι ιδιότητες για τη ForceAtlas είναι οι εξής:

  - **Αδράνεια (Inertia)** - Διατήρηση της ταχύτητας του κόμβου σε κάθε νέα επανάληψη.
  - **Δύναμη απομάκρυνσης (Repulsion strength)** - Πόσο σθεναρά κάθε κόμβος απορρίπτει τους υπόλοιπους.
  - **Δύναμη έλξης (Attraction strength)** - Πόσο έντονα κάθε ζεύγος συνδεδεμένων κόμβων προσελκύει ο ένας τον άλλον.

- **Μέγιστη μετατόπιση (Maximum displacement)** - Περιορίζει τη μετατόπιση κάθε κόμβου (για να αποτρέψει την υπερβολική απόρριψη όταν οι κόμβοι είναι πολύ κοντά).
- **Λειτουργία αυτόματης σταθεροποίησης (Auto stabilize function)** - Ενεργοποιεί την ψύξη των ασταθών κόμβων. Παρά την απώλεια αποτελεσματικότητας, αποτρέπει τους περισσότερους κόμβους από το να τρεμοπαίζουν.
- **Δύναμη αυτόματης σταθεροποίησης (Autostab strength)** - Ισχύς της λειτουργίας αυτόματης σταθεροποίησης.
- **Ευαισθησία αυτόματης σταθεροποίησης (Autostab sensitibvity)** - [0,1] Αντιπροσωπεύουν την αυτόματη προσαρμογή της λειτουργίας αντιτρεμοπαίγματος (στην πραγματικότητα, η αδράνεια ψυξης).
- **Βαρύτητα (Gravity)** - Αυτή η δύναμη προσελκύει όλους τους κόμβους στο κέντρο για να αποφευχθεί η διασπορά αποσυνδεδεμένων στοιχείων.
- **Διανομή έλξης (Attraction distribution)** - Η δύναμη έλξης κατανέμεται κατά μήκος των εξερχόμενων ζεύξεων. Αυτό τείνει να ωθήσει τους κόμβους στην περιφέρεια και να θέσει τις αρχές πιο κεντρικά.
- **Προσαρμογή κατά μέγεθος (Adjust by sizes)** - Αποφεύγει την επικάλυψη των κόμβων (ανάλογα με το μέγεθος κάθε κόμβου).
- **Ταχύτητα (Speed)** - Τιμή> 0, προεπιλογή 1. Επιτρέπει να αυξήσουμε την ταχύτητα σύγκλισης έναντι απώλεια ακριβείας.

- **ForceAtlas2**

Διάταξη ποιότητας: ένα γραμμικό μοντέλο έλξης γραμμικής απόπτωσης με λίγες προσεγγίσεις (BarnesHut). Η ταχύτητα υπολογίζεται αυτόματα. Παρακάτω βλέπουμε τις ιδιότητες για το ForceAtlas2:

- **Αριθμός νημάτων (Threads number)** - Περισσότερα νήματα σημαίνει μεγαλύτερη ταχύτητα αν οι πυρήνες μπορούν να το χειριστούν.
- **Ανοχή (Tolerance)** - Πόση ταλάντωση επιτρέπεται. Πάνω από 1 αποθαρρύνεται. Κάτω του ενός, μειώνει τη ταχύτητα έναντι περισσότερης ακρίβειας.

- **Κατά προσέγγιση απόρριψη (Approximate repulsion)** - Βελτιστοποίηση του Barnes Hut: πολυπλοκότητα προς n.ln (n), επιτρέπει μεγαλύτερα γραφήματα.
- **Προσέγγιση (Approximation)** - Θήτα(Theta) της βελτιστοποίησης του Barnes Hut.
- **Κλιμάκωση (Scaling)** - Πόση απώθηση θέλουμε. Όσο μεγαλύτερη η τιμή της τόσο μειώνεται ο θόρυβος στο γράφο.
- **Ισχυρότερη βαρύτητα (Stronger gravity)** - Ένας ισχυρότερος νόμος βαρύτητας.
- **Βαρύτητα (Gravity)** - Έλξη των κόμβων στο κέντρο. Αποτρέπει την απομάκρυνση των στοιχείων.
- **Αποτροπή κόμβων (Dissuade hubs)** - Διανομή έλξης κατά μήκος των εξωτερικών ακμών. Οι κόμβοι προσελκύονται λιγότερο και έτσι ωθούνται στα σύνορα.
- **Λειτουργία LinLog (LinLog mode)** - Αλλάζει το μοντέλο ForceAtlas από lin-lin σε lin-log (αφιέρωμα στον Andreas Noack). Κάνει τα υποσυστήματα πιο συμπαγή.
- **Αποτροπή επικαλύψεων (Prevent overlap)** - Χρήση μόνο όταν είναι χωροταξικά. Δεν θα πρέπει να χρησιμοποιείται μαζί με τη ” Κατά προσέγγιση απόρριψη ”.
- **Επιρροή βάρους ακμής (Edge weight influence)** - Πόση επιρροή δίνετε στο βάρος των ακμών. 0 είναι ”καμία επιρροή” και 1 είναι ”κανονική”.

- **Fruchterman Reingold**

Η διάταξη Fruchterman Reingold είναι ένας αλγόριθμος διάταξης, από το 1984. Έχει τρεις βασικές ιδιότητες:

- **Περιοχή (Area)** - Το μέγεθος της περιοχής του γραφήματος, για παράδειγμα 1000 για 100 κόμβους.
- **Βαρύτητα (Gravity)** - Η δύναμη έλξης όλων των κόμβων στο κέντρο για να αποφευχθεί η διασπορά αποσυνδεδεμένων στοιχείων.

  - **Ταχύτητα (Speed)** - Τιμή> 0 προεπιλογή 1, Αυξάνει την ταχύτητα σύγκλισης έναντι απώλειας ακριβείας.

- **Προσαρμογή επιγραφής (Label Adjust)**

  Η προσαρμογή επιγραφής λειτουργεί βάσει του μεγέθους κειμένου, απομακρύνοντας τους κόμβους και επομένως κάνει κάθε επιγραφή αναγνώσιμη. Οι ιδιότητες που μπορεί να προσαρμόσει ο χρήστης είναι :

  - **Ταχύτητα (Speed)** - Συντελεστής ταχύτητας.
  - **Συμπερίληψη μεγέθους κόμβου (Include Node size)** - Συμπεριλαμβάνει το μέγεθος του κόμβου στην απόρριψη.

- **Μη-επικάλυψη (Noverlap)**

  Απλά μια δύναμη απωθήσεως για να αποφευχθεί η επικάλυψη των κόμβων (όχι όμως για τις επιγραφές). Οι ιδιότητες της είναι οι παρακάτω:

  - **Ταχύτητα (Speed)** - Συντελεστής ταχύτητας.
  - **Αναλογία (Ratio)** - Η αναλογία της διασποράς των κόμβων σε σχέση με το μέγεθος τους.
  - **Περιθώριο (Margin)** - Η ελάχιστη απόσταση ανάμεσα στους κόμβους.

- **OpenOrd**

  Αλγόριθμος διάταξης για μεγάλης κλίμακας μη-κατευθυνόμενα γραφήματα. Μπορεί να κλιμακωθεί σε πάνω από 1 εκατομμύριο κόμβους, καθιστώντας τον ιδανικό για μεγάλα γραφήματα. Ωστόσο, μικρά γραφήματα (εκατοντάδες ή λιγότερα) δεν καταλήγουν πάντα να φαίνονται τόσο καλά. Αυτός ο αλγόριθμος αναμένει μη-κατευθυνόμενα σταθμισμένα γραφήματα και στοχεύει στην καλύτερη διάκριση των υποσυστημάτων.

  Οι ιδιότητές του χωρίζονται σε δύο κατηγορίες. Η πρώτη αντιπροσωπεύει το ποσοστό του χρόνου τον οποίο θα αφιερώσει στο κάθε στάδιο από τα παρακάτω: **Liquid**, **Expansion**, **Cooldown**, **Crunch**, **Simmer**.Η δεύτερη περιέχει τις εξής γενικές ιδιότητες:

  - **Περικοπή ακμής (Edge cut)** - 0 σημαίνει μη κοπή και 1 μέγιστη κοπή. Μεγαλύτερη κοπή σημαίνει ένα πιο συγκεντρωμένο αποτέλεσμα.
  - **Αριθμός νημάτων (Num threads)** - Ο αριθμός των νημάτων που θα χρησιμοποιηθεί για την εκτέλεση του αλγορίθμου. Αυξήστε αυτόν τον αριθμό για υπολογιστές πολλαπλών πυρήνων. Συνιστάται να τοποθετήσετε τον αριθμό του πυρήνα μείον ένα για να διατηρήσετε ένα νήμα για τη προβολή.
  - **Αριθμός επαναλήψεων (Num iterations)** - Αυξήστε τον αριθμό μόνο για πολύ μεγάλα γραφήματα. Περισσότερες επαναλήψεις ισούνται με περισσότερο χρόνο που χρειάζεται και λιγότερο πυκνό αποτέλεσμα. Το ελάχιστο είναι 100 επαναλήψεις και η προεπιλογή είναι 750.
  - **Σταθερός χρόνος (Fixed time)** - Όταν ορισμένοι κόμβοι είναι σταθεροί ρυθμίζει τη χρονική στιγμή που οι σταθεροί κόμβοι δεν θα μετακινηθούν. 0 σημαίνει ότι δεν θα σταθεροποιηθούν και ότι θα παραμείνουν σταθεροί.
  - **Τυχαίος σπόρος (Random seed)** - Το αποτέλεσμα του αλγορίθμου εξαρτάται από τον σπόρο, τον αριθμό των επαναλήψεων και τον αριθμό των νημάτων.

- **Τυχαία διάταξη (Random Layout)** Μια τυχαία κατανομή των κόμβων. Έχει μόνο μία ιδιότητα το:
  - **Μέγεθος χώρου (Space size)** - Το μέγεθος του χώρου για την τυχαία διανομή των κόμβων.

- **Περιστροφή (Rotate)**

  Περιστρέφει το γράφημα βάσει μοιρών:
  - **Γωνία (Angle)** - Δεξιόστροφη γωνία περιστροφής σε μοίρες (μπορεί να είναι αρνητική για αριστερόστροφη φορά).

- **Yifan Hu**

  Το αρχικό πρότυπο έλξης-απομάκρυνσης του Yifan Hu. Μειώνει το υπολογιστικό κόστος περιορίζοντας τον υπολογισμό της δύναμης στη ”γειτονιά”. Ο αλγόριθμος σταματάει μόνος του, καθώς έχει ένα προσαρμοστικό σύστημα ψύξης. Οι ιδιότητες του είναι οι παρακάτω:

- **Βέλτιστη απόσταση (Optimal distance)** - Το φυσικό μήκος των ελατηρίων. Οι μεγαλύτερες τιμές σημαίνουν ότι οι κόμβοι θα απέχουν πολύ περισσότερο.
- **Σχετική αντοχή (Relative strength)** - Η σχετική αντοχή μεταξύ ηλεκτρικής δύναμης (απομάκρυνσης) και δύναμης ελατηρίου (έλξη).
- **Αρχικό μέγεθος βήματος (Initial step size)** - Το αρχικό μέγεθος βήματος που χρησιμοποιήθηκε στη φάση ολοκλήρωσης. Ορίστε αυτήν την τιμή σε ένα σημαντικό μέγεθος σε σύγκριση με τη βέλτιστη απόσταση (10% είναι ένα καλό σημείο εκκίνησης).
- **Αναλογία βημάτων (Step ratio)** - Ο λόγος που χρησιμοποιείται για την ενημέρωση του μεγέθους βήματος σε επαναλήψεις.
- **Προσαρμοσμένη ψύξη (Adaptive cooling)** - Ελέγχει τη χρήση της προσαρμοσμένης ψύξης. Χρησιμοποιείται για να βοηθήσει τον αλγόριθμο διάταξης ώστε να αποφευχθούν τοπικά ελάχιστα ενέργειας.
- **Όριο σύγκλισης (Convergence threshold)** - Όριο σχετικής ενεργειακής σύγκλισης. Μικρότερες τιμές σημαίνουν μεγαλύτερη ακρίβεια.
- **Μέγιστο Επίπεδο Quadtree (Quadtree max level)** - Το μέγιστο επίπεδο που θα χρησιμοποιηθεί στην παράσταση quadtree. Μεγαλύτερες τιμές σημαίνουν μεγαλύτερη ακρίβεια.
- **Θήτα (Theta)** - Η παράμετρος theta για τα κριτήρια ανοίγματος Barnes-Hut. Μικρότερες τιμές σημαίνουν μεγαλύτερη ακρίβεια.

- **Yifan Hu Proportional**

  Τροποποιημένη έκδοση του Yifan Hu που χρησιμοποιεί το σύστημα αναλογικής μετατόπισης. Οι ιδιότητες της είναι ίδιες με τις Yifan Hu που αναφέραμε παραπάνω.

Ο πίνακας 4.1 παρουσιάζει μία ποσοτική σύγκριση της ποιότητας και της ταχύτητας των διαθέσιμων αλγορίθμων που βασίζονται στη δομή του γραφήματος και των σχέσεων ανάμεσα στα στοιχεία του. Για τη συμπλήρωση του πίνακα χρησιμοποιήσαμε τις πληροφορίες που μας παρέχει η Gephi για τους διαθέσιμους αλγόριθμους διάταξης,

τόσο στο ομώνυμο πρόγραμμα όσο και στο API της. Να σημειώσουμε ότι οι παραπάνω αλγόριθμοι μπορούν να συνδυαστούν με εκείνους που ασχολούνται μόνο με τα χαρακτηριστικά τους στο χώρο, όπως οι αλγόριθμοι διάταξης: Συστολή, Διαστολή, Προσαρμογή επιγραφής, Μη-επικάλυψη και Περιστροφή για ένα πιο ευανάγνωστο και καλαίσθητο αποτέλεσμα. Επίσης προτιμούμε να εφαρμόσουμε τη διάταξη υποσυστημάτων μετά από ένα αλγόριθμο του πίνακα 4.1 για την πιο ορθή τοποθέτηση των κόμβων στο χώρο.

Πίνακας 4.1: Σύγκριση αλγορίθμων διάταξης.

Αλγόριθμος	**Ποιότητα**	**Ταχύτητα**
ForceAtlas	★★★★★	★★★☆☆
ForceAtlas2	★★★★☆	★★★★☆
Fruchterman Reingold	★★☆☆☆	★★★☆☆
OpenOrd	★★★☆☆	★★★★★
Yifan Hu	★★★☆☆	★★★★☆
Yifan Hu Proportional	★★★☆☆	★★★★☆

4.6 Υπηρεσία φιλτραρίσματος

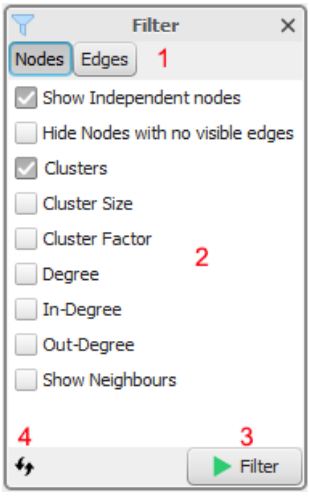
Η υπηρεσία φιλτραρίσματος μας επιτρέπει να διαλέξουμε τα στοιχεία του γράφου που θα εμφανίζονται ανάλογα με τις ιδιότητες του. Έχουμε δύο βασικές κατηγορίες φίλτρων, τα φίλτρα για τους κόμβους και τα φίλτρα για τις ακμές, τα οποία μπορούμε να τα χρησιμοποιήσουμε και συνδυαστικά. Η υπηρεσία φιλτραρίσματος είναι διαθέσιμη στο χρήστη μέσω του παράθυρου φιλτραρίσματος (Filter), το οποίο απεικονίζεται στο διάγραμμα 4.14. Οι βασικές περιοχές του παραθύρου είναι οι εξής:

1. Εμφανίζει τα διαθέσιμα φίλτρα για την κατηγορία που επιλέχθηκε (φίλτρα κόμβων ή ακμών).

2. Η περιοχή εμφάνισης και επιλογής διαθέσιμων φίλτρων.

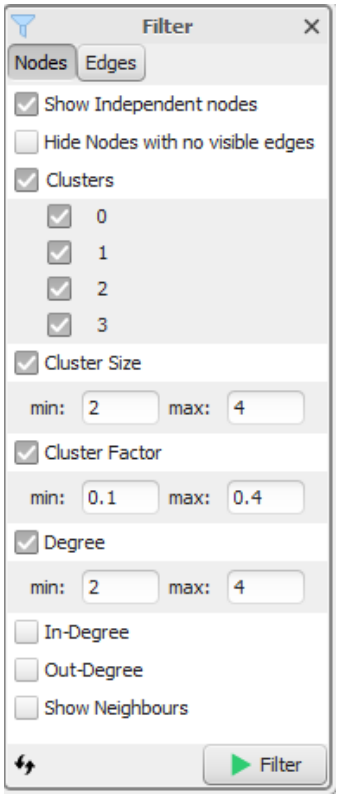
3. Το κουμπί φιλτραρίσματος, εφαρμόζει τα φίλτρα που επιλέχθηκαν για τους κόμβους και τις ακμές.

4. Το κουμπί επαναφοράς, επαναφέρει όλα τα φίλτρα στην προεπιλογή τους και εμφανίζει ολόκληρο το γράφο.



Διάγραμμα 4.14: Παράθυρο φιλτραρίσματος

4.6.1 Φιλτράρισμα κόμβων



Διάγραμμα 4.15: Φίλτρα κόμβων

Σε αυτή την ενότητα θα περιγράψουμε τα διαθέσιμα φίλτρα κόμβων, θα αναλύσουμε τη χρησιμότητα του καθένα και θα αναφέρουμε τον τρόπο χρήσης τους. Το σύνολο αυτών των φίλτρων είναι εννιά και είναι τα παρακάτω:

- **Εμφάνιση ανεξάρτητων κόμβων (Show independent nodes)** - Εμφανίζει ή αποκρύβει αντίστοιχα τους κόμβους που δεν έχουν καθόλου ακμές.
- **Απόκρυψη κόμβων με μη-εμφανείς ακμές (Hide nodes with no visible edges)** - Αποκρύβει τους κόμβους που έχουν ακμές αλλά δεν εμφανίζονται λόγω εφαρμογής φίλτρου ακμών. Δουλεύει συνδυαστικά με τα φίλτρα ακμών, αν δεν έχει προηγηθεί φιλτράρισμα στις ακμές δεν κάνει τίποτα.
- **Φίλτρο υποσυστημάτων (Clusters)** - Εμφανίζει τα επιλεγμένα υποσυστήματα, ενώ αποκρύβει τα μη-μαρκαρισμένα. Παρέχει μία λίστα με όλα τα υποσυστήματα του γράφου - για την εμφάνιση όλων των υποσυστημάτων, πρέπει να είναι όλα επιλεγμένα, όπως άλλωστε είναι και η προεπιλογή τους .

- **Φιλτράρισμα βάσει του μεγέθους υποσυστήματος (Cluster Size)** - Αποκρύβει τα υποσυστήματα με πλήθος κόμβων έξω από το πεδίο ορισμού επιλογής μας. Μπορούμε να συμπληρώσουμε ένα ή και τα δύο άκρα του πεδίου ορισμού, ελάχιστο (min) και μέγιστο (max). Η προεπιλογή τους είναι - ∞ και $+\infty$.

- **Φιλτράρισμα βάσει το συντελεστή υποσυστήματος (Cluster Factor)** - Αποκρύβει τα υποσυστήματα με συντελεστή υποσυστήματος έξω από το πεδίο ορισμού επιλογής μας. Μπορούμε να συμπληρώσουμε ένα ή και τα δύο άκρα του πεδίου ορισμού, ελάχιστο (min) και μέγιστο (max). Παίρνει τιμές στο πεδίο ορισμού $[0, 1]$.

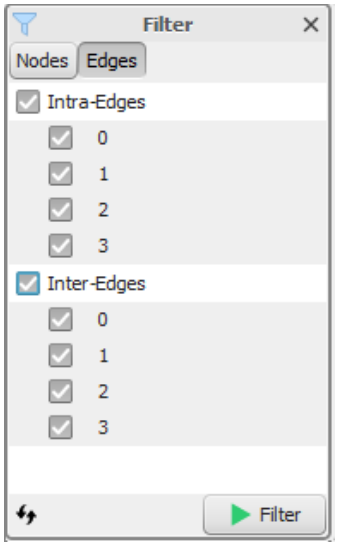
- **Φιλτράρισμα βάσει το συνολικό βαθμό(Degree)** - Αποκρύβει τους κόμβους με πλήθος ακμών που διέρχονται από αυτούς έξω από το πεδίο ορισμού επιλογής μας. Μπορούμε να συμπληρώσουμε ένα ή και τα δύο άκρα του πεδίου ορισμού, ελάχιστο (min) και μέγιστο (max). Η προεπιλογή τους είναι - ∞ και $+\infty$.

- **Φιλτράρισμα βάσει τον εσωτερικό βαθμό (In-Degree)** - Αποκρύβει τους κόμβους με πλήθος ακμών που τερματίζουν σε αυτούς έξω από το πεδίο ορισμού επιλογής μας. Μπορούμε να συμπληρώσουμε ένα ή και τα δύο άκρα του πεδίου ορισμού, ελάχιστο (min) και μέγιστο (max). Η προεπιλογή τους είναι - ∞ και $+\infty$.

- **Φιλτράρισμα βάσει το εξωτερικό βαθμό (Out-Degree)** - Αποκρύβει τους κόμβους με πλήθος ακμών που ξεκινούν από αυτούς έξω από το πεδίο ορισμού επιλογής μας. Μπορούμε να συμπληρώσουμε ένα ή και τα δύο άκρα του πεδίου ορισμού, ελάχιστο (min) και μέγιστο (max). Η προεπιλογή τους είναι - ∞ και $+\infty$.

- **Εμφάνιση γειτόνων (Show neighbors)** - Χρησιμοποιείται συνδυαστικά με άλλα φίλτρα. Εμφανίζει τους κόμβους πού συνδέονται από ακμές με τους κόμβους που τηρούν τα υπόλοιπα κριτήρια φιλτραρίσματος.

4.6.2 Φιλτράρισμα ακμών



Διάγραμμα 4.16: Φίλτρα ακμών

Σε αντίθεση με τα φίλτρα κόμβων, τα φίλτρα ακμών είναι πολύ λιγότερα, μόλις δύο. Οι ακμές άλλωστε δεν έχουν τόσα χαρακτηριστικά όσα οι κόμβοι. Μπορούν να χρησιμοποιηθούν όμως συνδυαστικά με εκείνα των κόμβων, ώστε να δημιουργήσουν χρήσιμες όψεις του γράφου. Τα δύο διαθέσιμα φίλτρα ακμών έχουν και τα δύο την ίδια μορφή, παρέχουν μία λίστα με όλα τα υποσυστήματα και από αυτά επιλεγούμε ποιου οι ακμές θα εμφανίζονται σύμφωνα με το φίλτρο. Τα δύο αυτά φίλτρα λοιπόν είναι:

- **Intra-Edges** - Εμφανίζει όλες τις intra-edges (ακμές με πηγαίο και στόχο κόμβο του ίδιου υποσυστήματος) των επιλεγμένων υποσυστημάτων, ενώ αποκρύβει εκείνες των μη-επιλεγμένων.
- **Inter-Edges** - Εμφανίζει όλες τις inter-edges (ακμές με πηγαίο και στόχο κόμβο διαφορετικού υποσυστήματος) των επιλεγμένων υποσυστημάτων, ενώ αποκρύβει εκείνες των μη-επιλεγμένων.

4.6.3 Παραδείγματα χρήσης συνδυασμών φίλτρων

Έχοντας ήδη περιγράψει τα διαθέσιμα φίλτρα ξεχωριστά, ας δούμε μερικά σενάρια χρήσης τους συνδυαστικά μεταξύ τους ή με άλλες υπηρεσίες. Παρακάτω θα αναφέρουμε μερικά παραδείγματα από όψεις που μπορούμε να δημιουργήσουμε χρησιμοποιώντας την υπηρεσία φιλτραρίσματος και το σκοπό που εξυπηρετούν αυτά τα παραδείγματα χρήσης.

Για την δημιουργία μίας όψης που δίνει έμφαση στην εξωτερική συνδεσιμότητα των υποσυστημάτων μπορούμε να αποκρύψουμε όλες τις εσωτερικές ακμές από το φίλτρο των *intraedges* για όλα τα υποσυστήματα και χρησιμοποιώντας το φίλτρο κόμβων *απόκρυψης κόμβων χωρίς εμφανείς ακμές (Hide nodes with no visible edges)*, να αποκρύψουμε τους κόμβους που έχουν μόνο εσωτερικές ακμές μέσα στο υποσύστημα και επομένως δε μας ενδιαφέρουν. Αυτή η όψη είναι εναλλακτική της λειτουργίας *Group*

της υπηρεσίας οπτικοποίησης και μας δίνει τη δυνατότητα να δούμε τους πραγματικούς κόμβους που συνδέονται με άλλα υποσυστήματα.

Για να εξετάσουμε τη συνδεσιμότητα ενός υποσυστήματος ή μιας ομάδας υποσυστημάτων μπορούμε να επιλέξουμε μόνο τα υποσυστήματα που μας ενδιαφέρουν από το φίλτρο κόμβων για *υποσυστήματα (Cluster)*, αποκρύβοντας τα υπόλοιπα, και να εφαρμόσουμε επίσης το φίλτρο *εμφάνισης γειτόνων (Show neighbors)*. Αυτός ο συνδυασμός φίλτρων μπορεί να χρησιμοποιηθεί είτε στην αρχική κατάσταση του γράφου ώστε να μας αναδείξει τους κόμβους με τους οποίους συνδέεται εξωτερικά είτε εφόσον έχει εφαρμοσθεί η λειτουργία *Group* ώστε να μας αναδείξει τα υποσυστήματα στα οποία ανήκουν οι εν λόγω κόμβοι.

Για την ανάδειξη των ανεξάρτητων από το υπόλοιπο σύστημα υποσυστημάτων μπορούμε εφόσον έχουμε εφαρμόσει τη λειτουργία Group, να εφαρμόσουμε είτε το φίλτρο *συνολικού βαθμού (Degree)* με μέγιστη τιμή ίση με μηδέν είτε το φίλτρο *εξωτερικού βαθμού (Out-Degree)* με μέγιστη τιμή ίση με μηδέν. Στη πρώτη περίπτωση τα υποσυστήματα που θα παραμείνουν, είναι εκείνα που είναι ανεξάρτητα από το υποσύστημα και το υπόλοιπο σύστημα από εκείνα, ενώ στη δεύτερη εκείνα που είναι ανεξάρτητα από τα υπόλοιπα υποσυστήματα αλλά κάποια στοιχεία του συστήματος είναι εξαρτημένα από αυτά. Αντίστοιχη διαδικασία μπορούμε να ακολουθήσουμε με την εφαρμογή του φίλτρου *εσωτερικού βαθμού (In-Degree)* για την ανάδειξη υποσυστημάτων που κανένα στοιχείο του συστήματος δεν εξαρτάται από αυτά.

Τέλος, για την εξέταση της εσωτερικής συνδεσιμότητας των υποσυστημάτων μπορούμε να αποκρύψουμε της εξωτερικές ακμές από το φίλτρο ακμών *interedges*, τους κόμβους που έχουν μόνο εξωτερικές ακμές από το φίλτρο *απόκρυψης κόμβων χωρίς εμφανής ακμές* και να επιλέξουμε μία κατάλληλη τιμή για το φίλτρο *συντελεστή υποσυστήματος (Cluster Factor)*. Μία τιμή κοντά στο 0 ως μέγιστη θα αναδείξει υποσυστήματα με πιο χαλαρή εσωτερική συνδεσιμότητα έναντι της εξωτερικής του, ενώ μία τιμή κοντά στο 1 ως ελάχιστη θα αναδείξει πιο συμπαγή υποσυστήματα με μικρότερη εξωτερική συνδεσιμότητα.

Κεφάλαιο 5

Επικύρωση αποτελεσμάτων και Συμπεράσματα

Η διατήρηση και η κατανόηση της δομής του πηγαίου κώδικα γίνεται όλο και πιο δύσκολη επειδή το μέγεθος και η πολυπλοκότητα των σύγχρονων συστημάτων λογισμικού αυξάνεται. Αυτό σε συνδυασμό με συναφή προβλήματα, όπως η έλλειψη λεπτομερούς τεκμηρίωσης του σχεδιασμού και η περιορισμένη διαθεσιμότητα του αρχικού σχεδιασμού του συστήματος, προσθέτει περαιτέρω δυσκολίες στην εργασία των επαγγελματιών λογισμικού που προσπαθούν να κατανοήσουν τη δομή μεγάλων και πολύπλοκων συστημάτων. Η εφαρμογή τεχνικών και εργαλείων συσταδοποίησης σε συστήματα λογισμικού φαίνεται να αποτελεί ένα πολλά υποσχόμενο τρόπο υποστήριξης των σχεδιαστών λογισμικού, επιτρέποντας τους να δημιουργήσουν αφηρημένες όψεις της δομής του συστήματος.

Η συνεισφορά μας στην έρευνα λογισμικού περιλαμβάνει την ανάπτυξη ενός αλγόριθμου που επιτρέπει τη διαίρεση της στατικής δομής ενός συστήματος λογισμικού σε επιμέρους αφηρημένα υποσυστήματα σε συνδυασμό με ένα γραφικό περιβάλλον που υποστηρίζει την οπτικοποίηση των αποτελεσμάτων και τη δημιουργία όψεων τους σύμφωνα με τις παραμέτρους που ορίζει ο χρήστης, για την καλύτερη και πιο εύκολη κατανόηση τους . Βασιζόμενοι στον αλγόριθμο αναρρίχησης λόφων που περιγράφεται στο (Mitchell και Mancoridis 2006), αναπτύξαμε μία δική μας εκδοχή του πρωτοτύπου αλγόριθμου με σκοπό την μείωση του χρόνου εκτέλεσης για γράφους εκτεταμένου μεγέθους, σε πιο ρεαλιστικά χρονικά πλαίσια.

Στην παρακάτω ενότητα επικυρώνουμε τόσο την επίτευξη αυτού του στόχου όσο

και τη διατήρηση της ποιότητας κατάτμησης. Η προσέγγισή μας είναι πολύτιμη για τους διαχειριστές λογισμικού επειδή χωρίζει γρήγορα τη δομή ενός συστήματος λογισμικού χρησιμοποιώντας τις εξαρτήσεις πόρων που καθορίζονται στη στατική δομή του πηγαίου κώδικα και παράλληλα τους δίνει τη δυνατότητα οπτικοποίησης του γράφου σε ένα φιλικό προς το χρήστη γραφικό περιβάλλον.

Επίσης, το εργαλείο που αναπτύξαμε επιτρέπει την εξατομίκευση των αποτελεσμάτων σε ένα πλήθος αρχείων τα οποία ο χρήστης μπορεί να χρησιμοποιήσει αργότερα ως είσοδο σε άλλα εργαλεία επεξεργασίας γράφων, όπως εκείνο της Gephi, για την εκμετάλλευση περαιτέρω λειτουργιών που δεν υλοποιούνται στο δικό μας εργαλείο. Ενώ η υλοποίηση του εργαλείου καθιστά δυνατή και εύκολη την μελλοντική επέκταση του τόσο στο επίπεδο συσταδοποίησης, όπως περιγράψαμε στην ενότητα 3.3, όσο και στο επίπεδο οπτικοποίησης.

5.1 Επικύρωση αποτελεσμάτων

Η διαφορά του χρόνου εκτέλεσης του πρωτοτύπου αλγορίθμου αναρρίχησης λόφων στα δύο εργαλεία, Bunch και Sheaf, ήταν αμελητέα παρά την διαφορετική τους υλοποίηση. Ενώ ο πρωτότυπος αλγόριθμος του (Mitchell και Mancoridis 2006), παρουσιάζει εξαιρετική απόδοση ταχύτητας κατά την εκτέλεση του σε μικρούς γράφους, το ίδιο δεν ισχύει και για τους μεγαλύτερους. Ένα εργαλείο συσταδοποίησης λογισμικού έχει νόημα όταν είναι ικανό να παρουσιάσει αποτελέσματα για μεγάλα κα σύνθετα συστήματα. Με γνώμονα τα παραπάνω οδηγηθήκαμε στη δημιουργία μιας ελαφρώς προσαρμοσμένης εκδοχής του αλγόριθμου αναρρίχησης λόφων που περιγράφουμε αναλυτικότερα στην ενότητα 2.3.4.

Σε αυτήν την ενότητα παρουσιάζουμε τα αποτελέσματα της εκτέλεσης του προσαρμοσμένου αλγόριθμου αναρρίχησης λόφων που αναπτύξαμε έναντι του πρωτοτύπου. Η σύγκριση γίνεται ανάμεσα στα δύο χαρακτηριστικά που ενδιαφέρουν το χρήστη και μπορούν να παρασταθούν από αριθμητικές τιμές, την ποιότητα κατάτμησης και την ταχύτητα εκτέλεσης.

Για τη σύγκριση τους χρησιμοποιήσαμε τα αποτελέσματα που παρήχθησαν από την εκτέλεση των δύο αλγορίθμων πάνω σε ένα πλήθος γράφων. Εκτελέσαμε τον πρωτότυπο αλγόριθμο από το εργαλείο Bunch (Mancoridis, Mitchell, Chen κ.ά. 1999) που

ανέπτυξαν οι Mitchell και Mancoridis, ενώ το δικό μας αλγόριθμο από το εργαλείο Sheaf που αναπτύξαμε. Έχοντας ήδη συγκρίνει την δική μας υλοποίηση του πρωτότυπου αλγόριθμου με εκείνη του Bunch παρατηρήσαμε ότι η διαφορά στο χρόνο εκτέλεσης είναι αμελητέα και συνεπώς θεωρούμε ότι οι παρακάτω μετρήσεις είναι ανεξάρτητες της υλοποίησης.

Για την επικύρωση των αποτελεσμάτων χρησιμοποιήσαμε ένα σύνολο γράφων που παράχθηκαν τυχαία από ένα εργαλείο που αναπτύξαμε συγκεκριμένα για την αντικειμενική σύγκριση μεταξύ των δύο εργαλείων. Τόσο το πλήθος των κόμβων όσο και εκείνων των ακμών τους επιλέχθηκε τυχαία μέσα από το εκάστοτε πεδίο ορισμού που ορίσαμε. Για την πιο ορθή σύγκριση των δύο αλγορίθμων δημιουργήσαμε δέκα παραδείγματα γράφων με τυχαίο πληθυσμό κόμβων ανάμεσα στις τιμές 10-100, 100-1.000 και 1.000-10.000, για τη σύγκριση του αλγορίθμου ανάμεσα σε μικρούς, μεσαίους και μεγάλους γράφους. Τα αποτελέσματα σύγκρισης των παραπάνω γράφων παρουσιάζονται στους πίνακες 5.1, 5.2 και 5.3 αντίστοιχα.

Πίνακας 5.1: Σύγκριση αποτελεσμάτων μεταξύ Bunch και Shief για γράφους 10-100 κόμβων.

Κόμβοι	MQ		Χρόνος (ms)	
	Bunch	Shief	Bunch	Shief
5	1,066	1,066	42	3
19	2,578	2,578	32	21
28	3,827	3,911	30	26
39	4,812	4,481	90	54
41	4,683	4,683	144	309
52	7,024	7,074	130	142
64	7,814	7,717	727	144
78	8,859	9,058	516	385
85	10,272	10,654	913	223
97	11,364	11,364	2935	321

Όπως, παρατηρούμε από το πίνακα 5.1 ο προσαρμοσμένος αλγόριθμος υστερεί σε σχέση με την ικανότητα του πρωτότυπου να συγκλίνει για μικρούς γράφους. Αυτό συμβαίνει γιατί ενώ και οι δύο ξεκινάνε από τυχαίες διαμερίσεις, του γράφου, ο προσαρμοσμένος αλγόριθμος επιλέγει το σύνολο των τοπικών βέλτιστων κινήσεων (ανά κόμβο) ως επόμενη, σε αντίθεση με τον πρωτότυπο ο οποίος διαλέγει σε κάθε βήμα

τη βέλτιστη απ' όλο το σύνολο των κινήσεων. Αυτό το χαρακτηριστικό του προσαρμοσμένου αλγόριθμου του επιτρέπει να αυτοδιορθώνεται σε βάθος χρόνου, το οποίο όμως δεν είναι διαθέσιμο στους μικρούς γράφους. Το παραπάνω πρόβλημα λύνεται χρησιμοποιώντας ένα σύνολο από τυχαία διαμερισμένους γράφους, αυξάνοντας έτσι την πιθανότητα δημιουργίας μίας ''καλής'' αρχικής διαμέρισης του γράφου. Με αυτό το τρόπο εξασφαλίζουμε την εύρεση της βέλτιστης λύσης έναντι μιας αμελητέας χρονικής επιβάρυνσης. Χρησιμοποιώντας παράλληλα νήματα για την εκτέλεση του αλγορίθμου στο σύνολο του αρχικού πληθυσμού και έχοντας ως δεδομένο ότι ο χρόνος εκτέλεσης είναι της τάξης των millisecond, η χρονική επιβάρυνση δε γίνεται αντιληπτή από το χρήστη.

Πίνακας 5.2: Σύγκριση αποτελεσμάτων μεταξύ Bunch και Shief για γράφους 100-1.000 κόμβων.

Κόμβοι	MQ		Χρόνος (s)	
	Bunch	Shief	Bunch	Shief
107	12,451	12,447	1,451	0,377
225	25,746	25,802	72,3	1,507
374	41,563	41,369	1.520,922	5,099
397	45,961	45,999	379,399	4,053
457	51,002	51,049	558,02	13,025
576	62,526	62,997	1.221,68	14,735
630	69,464	70,160	1.461,448	14,447
776	85,661	85,406	10.274,978	23,680
840	91,983	92,436	6.997,272	36,664
915	99,581	99,829	13.079,938	50,032

Όσο μεγαλώνει η έκταση των γράφων γίνεται πιο αντιληπτή η διαφορά ανάμεσα στην χρονική απόδοση των δύο αλγορίθμων. Τα αποτελέσματα του πίνακα 5.2 επιβεβαιώνουν την μείωση του χρόνου εκτέλεσης από 3,8% μέχρι και 260%, ανάλογα με την αύξηση των στοιχείων του γράφου. Βέβαια σημαντικός παράγοντας παραμένει η αρχική διαμέριση του γράφου στις οποίες ίσως και να οφείλονται μερικές διακυμάνσεις τόσο στην τιμή της ποιότητας κατάτμησης όσο και στο χρόνο εκτέλεσης των δύο αλγορίθμων.

Για το πίνακα 5.3 χρησιμοποιήσαμε την Incremental TurboMQ του εργαλείου

Bunch, η συμβατική TurboMQ δεν ήταν ικανή να τερματίσει σε ρεαλιστικά χρονικά πλαίσια, ξεπερνώντας τις δέκα ώρες για τα περισσότερα παραδείγματα.

Παρόλο όμως της χρήσης μιας πιο γρήγορης συνάρτησης παρατηρούμε ότι ο προσαρμοσμένος αλγόριθμος παρουσιάζει καλύτερα αποτελέσματα τόσο στο χρόνο εκτέλεσης όσο και στη ποιότητα κατάτμησης, αναδεικνύοντας με αυτό το τρόπο την ικανότητα αυτο-διόρθωσης του. Η μείωση του χρόνου εκτέλεσης πλέων κυμαίνεται από 1,17% εώς και 19,9%.

Πίνακας 5.3: Σύγκριση αποτελεσμάτων μεταξύ Bunch και Shief για γράφους 1.000-10.000 κόμβων.

Κόμβοι	MQ		Χρόνος (s)	
	Bunch	Shief	Bunch	Shief
1487	374,172	374,381	21,858	18,571
2794	698,545	699,626	213,195	90,527
3742	933,215	933,912	526.518	175,793
4882	1.216,773	1.217,451	2.895,641	242,646
5425	1.354,982	1.355,773	3.981,233	350,345
6736	1.680,194	1.682,005	4.434,984	518,153
7337	1.829,958	1.831,238	6.376,854	574,856
8215	2.056,871	2.057,199	8.614,395	739,988
8873	2.212,844	2.214,907	12.847,856	929,765
9761	2.434,756	2.435,919	19.978,968	1.009,099

5.2 Καταληκτικές παρατηρήσεις

Αυτή η εργασία συμβάλλει στον τομέα της αντίστροφης μηχανικής αναπτύσσοντας νέες τεχνικές αναζήτησης που απευθύνονται στο πρόβλημα συσταδοποίησης λογισμικού. Έχουμε αναπτύξει ένα ολοκληρωμένο εργαλείο που εφαρμόζει τον αλγόριθμο μας, μαζί με πρόσθετες υπηρεσίες για την υποστήριξη της οπτικοποίησης. Συχνά, τα κίνητρα που χρησιμοποιούνται από τους ερευνητές για να περιγράψουν τα οφέλη της συσταδοποίησης λογισμικού περιλαμβάνουν την απλούστευση της συντήρησης λογισμικού και τη βελτίωση της κατανόησης του προγράμματος. Έτσι, ελπίζουμε ότι αυτή η εργασία θα βοηθήσει στην προώθηση της χρήσης τεχνολογιών συσταδοποίησης λογισμικού εκτός του ερευνητικού περιβάλλοντος, προωθώντας το ενδιαφέρον των προ-

γραμματιστών προς τα εργαλεία μηχανικής λογισμικού.

Η συνεισφορά αυτής της εργασίας συνίσταται στη βελτίωση του αλγορίθμου που παρουσιάζεται στο (Mitchell και Mancoridis 2006) ώστε να μπορεί να δουλέψει ο αλγόριθμος για προγράμματα ρεαλιστικού μεγέθους και στην ανάπτυξη μιας εφαρμογής που επιτρέπει την οπτικοποίηση των αποτελεσμάτων αυτών. Με αυτή την εργασία στοχεύουμε να βοηθήσουμε τους επαγγελματίες μηχανικούς λογισμικού, τους εκπαιδευτικούς και άλλους ερευνητές, δημιουργώντας ένα εργαλείο που παρέχει ένα απλό και εύκολα προσβάσιμο περιβάλλον χρήστη. Ο πηγαίος κώδικας της εφαρμογής Sheaf που αναπτύξαμε είναι διαθέσιμος στον ιστότοπο `https://bitbucket.org/GeorgiaGrigor/clusteringtool`.

Βιβλιογραφία

Bastian, Mathieu, Sebastien Heymann, Mathieu Jacomy κ.ά. (2009). «Gephi: an open source software for exploring and manipulating networks.» Στο: ICWSM 8, σσ. 361–362.

Doval, Diego, Spiros Mancoridis και Brian S Mitchell (1999). «Automatic clustering of software systems using a genetic algorithm». Στο: Software Technology and Engineering Practice, 1999. STEP'99. Proceedings. IEEE, σσ. 73–81.

Harman, Mark και Bryan F Jones (2001). «Search-based software engineering». Στο: Information and software Technology 43.14, σσ. 833–839.

Harman, Mark, Phil McMinn κ.ά. (2012). «Search based software engineering: Techniques, taxonomy, tutorial». Στο: Empirical software engineering and verification. Springer, σσ. 1–59.

Mancoridis, Spiros, Brian S Mitchell, Yihfarn Chen κ.ά. (1999). «Bunch: A clustering tool for the recovery and maintenance of software system structures». Στο: Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on. IEEE, σσ. 50–59.

Mancoridis, Spiros, Brian S Mitchell, Chris Rorres κ.ά. (1998). «Using automatic clustering to produce high-level system organizations of source code». Στο: Program Comprehension, 1998. IWPC'98. Proceedings., 6th International Workshop on. IEEE, σσ. 45–52.

Mitchell, Brian S και Spiros Mancoridis (2001). «Comparing the decompositions produced by software clustering algorithms using similarity measurements». Στο: Software Maintenance, 2001. Proceedings. IEEE International Conference on. IEEE, σσ. 744–753.

— (2002a). A heuristic search approach to solving the software clustering problem. Drexel University Philadelphia, PA, USA.

Mitchell, Brian S και Spiros Mancoridis (2002b). «Using heuristic search techniques to extract design abstractions from source code». Στο: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Inc., σσ. 1375–1382.

— (2006). «On the automatic modularization of software systems using the bunch tool». Στο: IEEE Transactions on Software Engineering 32.3, σσ. 193–208.

Mitchell, Brian S, Spiros Mancoridis και Martin Traverso (2002). «Search based reverse engineering». Στο: Proceedings of the 14th international conference on Software engineering and knowledge engineering. ACM, σσ. 431–438.

Praditwong, Kata, Mark Harman και Xin Yao (2011). «Software module clustering as a multi-objective search problem». Στο: IEEE Transactions on Software Engineering 37.2, σσ. 264–282.

Wikipedia (2017a). Apache Maven. URL: https://en.wikipedia.org/wiki/Apache_Maven (visited on 08/25/2017).

— (2017b). Gephi. URL: https://en.wikipedia.org/wiki/Gephi (visited on 08/25/2017).

— (2017c). JUnit. URL: https://en.wikipedia.org/wiki/JUniti (visited on 08/25/2017).

— (2017d). Look and feel. URL: https://en.wikipedia.org/wiki/Look_and_feel (visited on 08/25/2017).

— (2017e). Model–view–controller. URL: https://en.wikipedia.org/wiki/Model-view-controller (visited on 08/25/2017).

— (2017f). Swing (Java). URL: https://en.wikipedia.org/wiki/Swing_(Java) (visited on 08/25/2017).

Μιχαηλίδης, Μιχαήλ (2017). Στατική Ανάλυση Προγραμμάτων SQL με σκοπό την κατανόηση του κώδικα.

Η εργασία αυτή στοιχειοθετήθηκε με το πρόγραμμα XeLaTeX. Για τη στοιχειοθέτηση της βιβλιογραφίας χρησιμοποιήθηκε το πρόγραμμα `biber` και `biblatex`. Οι γραμματοσειρές που χρησιμοποιήθηκαν είναι οι Times New Roman και `Courier New`.