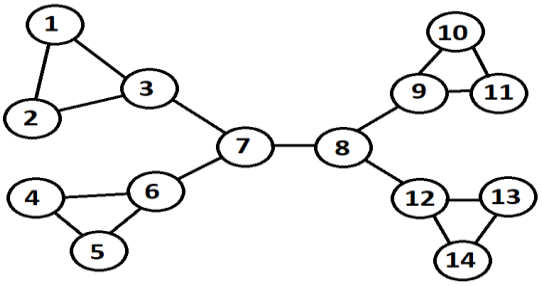
Αλγόριθμοι για Δεδομένα Ευρείας Κλίμακας

LAB2 Report:

Κουφόπουλος Μύρων : 4398
Χριστόπουλος Κωνσταντίνος : 4527

Θέμα της εργασίας αυτής είναι η ανίχνευση κοινοτήτων σε γραφήματα κοινωνικών δικτύων, μέσω τεχνικών συσταδοποίησης. Πιο συγκεκριμένα θα υλοποιήσουμε ιεραρχικές τεχνικές συσταδοποίησης και θα υπολογίσουμε την βέλτιστη διαμέριση του γραφήματος σε κοινότητες, βάσει της μετρικής αρθρωτότητας(modularity).
Στην εργασία, για να κατανοήσουμε και να επιβεβαιώσουμε τη λειτουργία των συναρτήσεων που δημιουργούμε χρησιμοποιήσαμε το παρακάτω γράφημα. Για αυτό το λόγο θα παρέχουμε στην εξήγηση κάθε συνάρτησης κατάλληλα παραδείγματα με βάση το συγκεκριμένο γράφημα.



Programming Task 1A:

Υλοποιούμε την συνάρτηση read_graph_from_csv() για να διαβάσουμε ολόκληρο το CSV αρχείο ακμών. Αρχικά διαβάζουμε το από το αρχείο και φορτώνουμε μόνο τις πρώτες NUM_LINKS ακμές. Στη συνέχεια, αφαιρούμε τις self-loop(A,A) ακμές τις οποίες θεωρούμε duplicates, με αυτό το τρόπο εξασφαλίζουμε πως η συνάρτηση from_pandas_edgelist() θα διαχειριστεί μόνο τις παράλληλες ακμές. Για τους κόμβους, διαβάζουμε μία-μία τις στήλες από το αρχείο ,τις αποθηκεύουμε σε λίστες και έπειτα τις συγχωνεύουμε σε μία λίστα(node_names_list). Μετά, δημιουργώντας ένα dictionary με κλειδιά την λίστα αυτή, διαγράφουμε τις διπλές εμφανίσεις κόμβων. Το dictionary το μετατρέπουμε σε λίστα και τέλος δημιουργούμε το γράφημα.

Programming Task 1B:

Σε αυτό το βήμα δίνεται η δυνατότητα στο χρήστη να αυξήσει τις ακμές του γραφήματος για να μειωθούν οι συνεκτικές συνιστώσες ή για να αποφευχθεί η ύπαρξη γεφυρών.

i. Υλοποιούμε την συνάρτηση add_random_edges_to_graph(), στην οποία προσθέτουμε ένα πλήθος από ακμές στο γράφημα προς τυχαίους κόμβους. Αρχικά, για κάθε κόμβο του γραφήματος βρίσκουμε τους μη γειτονικούς του κόμβους. Από αυτούς επιλέγουμε τυχαία έναν κόμβο στον οποίο θα προσπαθήσουμε NUM_RANDOM_EDGES φορές να προσθέσουμε μία ακμή, από τον αρχικό προς τον επιλεγμένο κόμβο. Σε κάθε προσπάθεια, θα αποφασίζεται, με ρίψη τυχαίου νομίσματος το οποίο θα έχει πιθανότητα επιτυχίας EDGE_ADDITION_PROBABILITY, αν θα προστεθεί στο γράφημα η νέα ακμή.
ii. Υλοποιούμε την συνάρτηση add_hamilton_cycle_to_graph(), στην οποία ‘εμφυτεύουμε’ έναν κύκλο Hamilton, δηλαδή δημιουργούμε έναν κύκλο που θα

περιλαμβάνει όλες τις κορυφές του γραφήματος. Πρώτα ελέγχουμε αν το γράφημα δεν συνεκτικό, δηλαδή αν περιέχει τουλάχιστον δύο συνεκτικές συνιστώσες, αν περιέχει μόνο μία το γράφημα είναι συνεκτικό και δεν δημιουργούμε κύκλο Hamilton. Στη συνέχεια, για την δημιουργία του κύκλου Hamilton παίρνουμε ακολουθιακά μία-μία τις κορυφές του γραφήματος (με τη σειρά που είναι αποθηκευμένες στη node_names_list) και προσθέτουμε ακμή από τον κόμβο που βρισκόμαστε προς τον επόμενο. Όταν βρεθούμε στο τελευταίο κόμβο, για να ολοκληρωθεί σωστά ο κύκλος Hamilton, προσθέτουμε ακμή από αυτόν προς τον αρχικό κόμβο.

Programming Task 1Γ:

Σε αυτό το βήμα, υλοποιούμε τον αλγόριθμο του Girvan-Newman (GN), ο οποίος αποσκοπεί στην δημιουργία Κ+1 κοινοτήτων, σε ένα γράφημα με Κ συνεκτικές συνιστώσες. Ακολουθήσαμε δυο διαφορετικές προσεγγίσεις υλοποίησης του αλγορίθμου:

i. Στην πρώτη προσέγγιση, χρησιμοποιήσαμε την έτοιμη συνάρτηση girvan_newman() από την βιβλιοθήκη της «networkx.algorithms.community.centrality», βρίσκοντας κάθε φορά τη μεγαλύτερη συνεκτική συνιστώσα (GCC) και διαμερίζοντάς την σε δυο υποκοινότητες (LC1, LC2). Έτσι, καταφέρνουμε να δημιουργήσουμε Κ+1 κοινότητες από ένα γράφημα Κ κοινοτήτων.
ii. Στη δεύτερη προσέγγιση, αναπτύξαμε τη δική μας υλοποίηση του GN. Αρχικά, υπολογίζουμε τις Κ συνεκτικές συνιστώσες του γραφήματος και τις ταξινομούμε σε μία λίστα σε φθίνουσα σειρά με αποτέλεσμα στην πρώτη θέση να βρίσκεται η μεγαλύτερη συνεκτική συνιστώσα (GCC). Θεωρούμε κάθε συνιστώσα ως μια κοινότητα στο γράφημά μας. Στην συνέχεια, θεωρούμε νέο γράφημα max_conn_component = G[LC] που είναι το υπογράφημα που επάγεται από τις κορυφές της μεγαλύτερης κοινότητας και εφαρμόζουμε τον αλγόριθμο του Girvan-Newman πάνω σε αυτό. Συγκεκριμένα, αφαιρούμε από αυτή διαδοχικά ακμές με τη μεγαλύτερη τιμή betweenness-cerntrality μέχρι να αποσυνδεθεί η συγκεκριμένη συνιστώσα σε δυο υποκοινότητες (LC1, LC2). Τέλος, αφαιρούμε από το community_tuples την LC και προσθέτουμε τις δυο υποκοινότητες που δημιουργήθηκαν από τη εκτέλεση του αλγορίθμου του GN με αποτέλεσμα να δημιουργούνται από γράφημά μας κάθε φορά Κ+1 συνεκτικές συνιστώσες.

Programming Task 1Δ:

Στη συγκεκριμένη περίπτωση, πραγματευόμαστε μια συγκεκριμένη μέθοδο, τη Διαιρετική Συσταδοποίηση. Αυτή η μέθοδος επιστρέφει την ιεραρχία διαμερίσεων των κοινοτήτων σε υποκοινότητες (hierarchy_of_community_tuples) . Συγκεκριμένα, ξεκινάμε έχοντας ακριβώς Κ κοινότητες μία για κάθε συνεκτική συνιστώσα του γραφήματος και «χωρίζουμε» τη μεγαλύτερη κοινότητα που έχουμε κάθε φορά σε δυο υποκοινότητες σύμφωνα με τον αλγόριθμο του Girvan-Newman που υλοποιήσαμε στο προηγούμενο βήμα. Τέλος, ο αλγόριθμος τερματίζεται όταν θα έχουν δημιουργηθεί number_of_divisions κοινότητες στο γράφημα.

Ως προς την υλοποίηση της συγκεκριμένης μεθόδου, αρχικά, αρχικοποιούμε τα community_tuples με βάση τις συνεκτικές συνιστώσες που έχουμε στο γράφημά μας. Έπειτα, δημιουργούμε ένα αντίγραφο του αρχικού γραφήματος G με σκοπό να μπορούμε να διαγράφουμε ακμές χωρίς να επηρεάζεται το G. Στη συνέχεια, επαναληπτικά μέχρι να δημιουργηθούν number_of_divisions κοινότητες, εντοπίζουμε την μεγαλύτερη συνεκτική συνιστώσα (LC), εφαρμόζουμε τον αλγόριθμο του Girvan-Newman σε αυτήν ο οποίος μας επιστρέφει τις δυο υποκοινότητες (LC1, LC2), που προκύπτουν από τη διαμέριση της μεγαλύτερης κοινότητας (LC). Ύστερα, προσθέτουμε στη λίστα hierarchy_of_community_tuples τη τριάδα από tuples [LC, LC1, LC2] και διαγράφουμε από τη λίστα community_tuples την LC, προσθέτοντας παράλληλα τις LC1 και LC2. Τέλος, στη μεγαλύτερη κοινότητα (LC), διαδοχικά αφαιρούμε ακμές με το υψηλότερο betweeness, έτσι ώστε η μέγιστη συνεκτική συνιστώσα (κοινότητα) να χωριστεί στις δυο αυτές υποκοινότητες που μας επέστρεψε ο αλγόριθμος του Girvan και στην επόμενη επανάληψη να εντοπιστεί μια διαφορετική μέγιστη συνεκτική συνιστώσα.

Programming Task 1E:

Σε αυτό το τελικό βήμα υλοποιούμε δυο διαφορετικές ρουτίνες, οι οποίες έχουν σκοπό την αποτύπωση του γραφήματος σε κοινότητες.

i. Υλοποιούμε τη συνάρτηση visualize_communities(), η οποία αποτυπώνει τις διαμερίσεις του γραφήματος σε κοινότητες. Έτσι, σύμφωνα με τις κοινότητες που έχουν δημιουργηθεί στη λίστα community_tuples, ανατίθεται ένα τυχαίο χρώμα ανά κοινότητα και τυπώνεται στην οθόνη το αρχικό γράφημα έχοντας διαφορετικά χρώματα στους κόμβους διαφορετικών κοινοτήτων.

ii. Στο συγκεκριμένο ερώτημα υλοποιούμε την συνάρτηση determine_opt_community_structure() η οποία για κάθε διαμέριση στην ιεραρχία χωριστά μετράει την τιμή αρθρωτότητας. Αρχικά βρίσκουμε πόσες κοινότητες έχουμε στο γράφημα μας μέσω των συνεκτικών συνιστωσών, με αυτό το τρόπο γνωρίζουμε κάθε φορά πόσες κοινότητες υπήρχαν στο γράφημα για κάθε modularity που υπολογίσαμε. Στη συνέχεια, δημιουργούμε ένα υπογράφημα του G από την πρώτη διαμέριση που έχουμε στην ιεραρχία των διαμερίσεων, αυτό το κάνουμε ώστε να μπορούμε να χρησιμοποιήσουμε την έτοιμη συνάρτηση modularity καθώς γνωρίζουμε πόσες κόμβους και πόσες ακμές περιείχε το γράφημα πριν την διαμέριση. Υπολογίζουμε την τιμή της αρθρωτότητας για τα δύο γραφήματα LC1 και LC2 που είναι αποθηκευμένα στην ιεραρχία των διαμερίσεων. Επαναλαμβάνουμε την συγκεκριμένη διαδικασία μέχρι να τελειώσουν οι διαμερίσεις που έχουν γίνει στο γράφημα. Στη πρώτη επανάληψη κρατάμε το partition που έχει σαν βέλτιστο και αν προκύψει κάποια τιμή αρθρωτότητας πιο μεγάλη δηλαδή πιο βέλτιστη, ενημερώνουμε το κατάλληλο πεδίο. Τέλος, δημιουργούμε ένα ραβδόγραμμα στο οποίο περνάμε τις τιμές αρθρωτότητας και το πλήθος των κοινοτήτων που είχαμε σε κάθε επανάληψη. Για το πλήθος κοινοτήτων, αφού γνωρίζουμε πόσες είχαμε στο αρχικό γράφημα G στην συνέχεια για κάθε μία τριάδα που υπάρχει στην ιεραρχία των κοινοτήτων αυξάνουμε κατά 1.