Ειδικά Θέματα Τεχνολογίας Λογισμικού

Έργο:

NetworkX

Ομάδα Υλοποίησης: Overriders
http://overriders.blogspot.com/

Καρακατσάνης Κωνσταντίνος (8120050)
Σωτηρόπουλος Θοδωρής (8120127)

Έκδοση Παραδοτέου: Τελική

8 Ιουνίου 2015, Αθήνα

Περιεχόμενα

1. Κατανόηση και τεκμηρίωση του έργου 3
2. Πλάτος αλλαγών 4
Προσθήκη αλγόριθμου Johnson για τα συντομότερα μονοπάτια 4
Ανανέωση του αλγόριθμου Johnson ώστε να μην μεταβάλλει το γράφο 5
Προσθήκη συναρτήσεων που ελέγχουν αν ένας γράφος έχει ακμές με βάρη και ακμές με αρνητικά βάρη. 5
Προσθήκη προσεγγιστικών αλγορίθμων για το πρόβλημα του περιοδεύοντος πωλητή. 6
Προσθήκη αλγορίθμων για την εύρεση κοινωνιών σε ένα γράφο. 7
Προσθήκη αλγορίθμου για την διάταξη των κόμβων 8
3. Ποιότητα υλοποίησης 9
4. Ολοκλήρωση 9
5. Έλεγχος 9
6. Συνεργασία με την ομάδα ανάπτυξης 10
7. Παρουσιάσεις 10
8. Οργάνωση στο GitHub (wiki, issues, commits, branching) 10

1. Κατανόηση και τεκμηρίωση του έργου

Το NetworkX είναι ένα πακέτο στην γλώσσα προγραμματισμού Python και αφορά την δημιουργία, μελέτη, ανάλυση και οπτικοποίηση γράφων. Πρόκειται για ένα έργο ανοιχτού λογισμικού γραμμένο σε Python εξ' ολοκλήρου (99.9%) και διανέμεται κάτω από την BSD άδεια. Λόγω του ότι η Python υποστηρίζει πολλές ιδεολογικές δομές και λόγω της φύσης του ίδιου του έργου, το NetworkX ακολουθεί πολλές διαφορετικές φιλοσοφίες προγραμματισμού, όπως είναι ο αντικειμενοστρεφής, συναρτησιακός, διαδικασιακός και συναρμολογούμενος προγραμματισμός.

Το NetworkX στον πυρήνα του χρησιμοποιεί, όσο πουθενά αλλού, την αντικειμενοστρέφεια ώστε να αναπαραστήσει τους γράφους. Το NetworkX υλοποιεί μια πλούσια συλλογή από αλγορίθμους που αφορούν έναν πολύ μεγάλο αριθμό προβλημάτων που συναντάμε σε γράφους. Αυτοί οι αλγόριθμοι είναι υλοποιημένοι σε συναρτήσεις, δηλαδή δεν ανήκουν σε κάποια κλάση, και οι όλες αυτές οι συναρτήσεις βασίζονται πάνω στον πυρήνα του NetworkX και είτε λαμβάνουν σαν όρισμα έναν αντικείμενο γράφου για να γίνει η μελέτη, ανάλυση και οπτικοποίησή του, είτε δημιουργούν ένα νέο αντικείμενο γράφου στην περίπτωση που πρόκειται για αλγορίθμους δημιουργίας γράφων, π.χ. τυχαίων γράφων.

Αυτές οι συναρτήσεις είναι ομαδοποιημένες σε modules, και τα modules σε πακέτα βάσει της κατηγορίας των αλγορίθμων που υλοποιούν. Για παράδειγμα, οι αλγόριθμοι που αφορούν τα συντομότερα μονοπάτια για γράφους με βάρη, βρίσκονται σε ένα module με όνομα *weighted.py* το οποίο ανήκει σε ένα πακέτο με όνομα *shortest_paths (*περιέχοντας και άλλα modules που υλοποιούν αλγορίθμους για το συγκεκριμένο πρόβλημα) το οποίο ανήκει σε ένα πακέτο με όνομα *algorithms* (περιέχοντας και άλλα υποπακέτα ή modules για άλλες κατηγορίες αλγορίθμων, όπως είναι για παράδειγμα, αλγόριθμοι για διμερείς γράφους ή για την εύρεση μετρικών σε κόμβους).

Με βάση τα παραπάνω, ένας χρήστης του NetworkX, μπορεί να δημιουργήσει ένα αντικείμενο γράφου, χρησιμοποιώντας συναρτήσεις δημιουργίας γράφων όπως είναι τυχαίοι γράφοι ή να διαβάσει για παράδειγμα ένα αρχείο που αναπαριστά ένα γράφο (*graphml, xml, txt, κτλ*) και να περάσει αυτό το αντικείμενο σε συναρτήσεις του NetworkX που υλοποιούν διάφορους αλγορίθμους ώστε να κάνει την ανάλυση και μελέτη του γράφου του. Αφού ο χρήστης φτάσει σε μια Α ανάλυση χρησιμοποιώντας το NetworkX, μπορεί να συνεχίσει την ανάλυση του γράφου του, είτε γράφοντας το δικό του κώδικα και υλοποιώντας τους δικούς του αλγορίθμους, είτε συνεχίζοντας την ανάλυσή του σε άλλο εργαλείο ανάλυσης γράφων εκμεταλλευόμενος συναρτήσεις του NetworkX, που του επιτρέπουν να κάνει εξαγωγή του γράφου σε κάποια μορφή (*graphml, pajek, xml, txt, κτλ*), ώστε να την περάσει ως input στο εργαλείο ανάλυσης όπου θα συνεχίσει την ανάλυσή του. Το NetworkX, λοιπόν μπορεί να χρησιμοποιηθεί αυτόνομα από μαθηματικούς, προγραμματιστές, κτλ, καθώς και να χρησιμοποιηθεί ως βάση για εφαρμογές ανάλυσης γράφων.

Ο κώδικάς του καλύπτεται με πάνω από 1800 unit tests (nosetests) και η κάλυψη ξεπερνάει το 90% (~92%). Όσον αφορά την τεκμηρίωσή του, αυτή είναι πολύ εκτενής και αναλυτική και μάλιστα ξεπερνάει τις 500 σελίδες. Για αυτόν τον λόγο, ο κώδικάς μας είχε επίσης αναλυτική και εκτενή επεξήγηση των όσων υλοποιήθηκαν.

2. Πλάτος αλλαγών

Στα πλαίσια της συνεισφοράς μας στο έργο (και λόγω της φύσης του έργου) αποφασίσαμε να κάνουμε μια αλγοριθμική συνεισφορά. Επομένως, υλοποιήσαμε μια σειρά αλγορίθμων για διάφορες κατηγορίες προβλημάτων που συναντιούνται σε γράφους.

Συγκεκριμένα υλοποιήσαμε αλγορίθμους για τα παρακάτω προβλήματα:

- Πρόβλημα Συντομότερων μονοπατιών
- Πρόβλημα Περιοδεύοντος Πωλητή
- Πρόβλημα Εύρεσης Κοινωνιών
- Πρόβλημα Διάταξης Κόμβων

Εκτός από την υλοποίηση αλγορίθμων που λύνουν τα παραπάνω προβλήματα, υλοποιήσαμε δύο συναρτήσεις για τον έλεγχο αν ένας γράφος έχει (αρνητικά) βάρη στις ακμές τους.

Ας πάρουμε κάθε ένα πρόβλημα ξεχωριστά.

Προσθήκη αλγόριθμου Johnson για τα συντομότερα μονοπάτια

Το πρόβλημα των συντομότερων μονοπατιών είναι από τα κλασικά προβλήματα που συναντάμε σε ένα γράφο. Έχουν δημοσιευτεί λοιπόν πολλοί αλγόριθμοι που λύνουν το συγκεκριμένο πρόβλημα. Το NetworkX υλοποιεί μια συλλογή τέτοιων αλγόριθμων όπως είναι ο αλγόριθμος του Dijkstra, Bellman-Ford, Floyd-Warshall, A*.
Ένας ακόμα αλγόριθμος που λύνει το συγκεκριμένο πρόβλημα είναι ο αλγόριθμος του Johnson, όποιος λύνει το πρόβλημα εύρεσης συντομότερων μονοπατιών μεταξύ όλων των ζευγαριών κόμβων ενός γράφου και λειτουργεί ακόμα και σε γράφους με αρνητικά βάρη. Το NetworkX δεν υλοποιούσε το συγκεκριμένο αλγόριθμο, επομένως σκεφτήκαμε ότι είναι μια καλή ιδέα να τον υλοποιήσουμε εμείς ώστε να προστεθεί στην συλλογή του. Και έτσι κάναμε. Δημιουργήσαμε μια νέα συνάρτηση για το συγκεκριμένο αλγόριθμο μαζί με τα tests και έτσι ανοίξαμε ένα νέο pull request το οποίο και έγινε merge μετά από 10 μέρες περίπου. Μπορείτε να το δείτε στο παρακάτω σύνδεσμο.

https://github.com/networkx/networkx/pull/1481

Επίσης παρακάτω φαίνεται το πώς μπορεί να χρησιμοποιηθεί η συνάρτηση που χρησιμοποιήσαμε με το όνομα «**johnson**», και τι αποτέλεσμα επιστρέφει.

```
Examples
--------
>>> import networkx as nx
>>> graph = nx.DiGraph()
>>> graph.add_weighted_edges_from([('0', '3', 3), ('0', '1', -5),
... ('0', '2', 2), ('1', '2', 4), ('2', '3', 1)])
>>> paths = nx.johnson(graph, weight='weight')
>>> paths['0']['2']
['0', '1', '2']
```

Ανανέωση του αλγόριθμου Johnson ώστε να μην μεταβάλλει το γράφο

Μετά την αποδοχή του αιτήματος αλλαγών που αφορούσε την προσθήκη συνάρτησης που υλοποιούσε τον αλγόριθμο Johnson, τέθηκε από ένα άτομο το ζήτημα ότι η συγκεκριμένη υλοποίηση μας μεταβάλλει το γράφο που περνιέται σαν όρισμα και αυτό μπορεί να οδηγήσει την αλλαγή της σειράς επανάληψης του λεξικού όπου αποθηκεύονται διάφορες πληροφορίες που αφορούν το γράφο όπως είναι κόμβοι και ακμές. Συνεπώς, θα έπρεπε να αντιμετωπίσουμε το συγκεκριμένο ζήτημα.

Επίσης, θα πρέπει να σημειωθεί ότι αλγόριθμος Johnson συνδυάζει τους αλγόριθμους Dijkstra και Bellman-Ford για να παράξει το αποτέλεσμά του. Χρησιμοποιεί τον αλγόριθμο Dijkstra αλλά με αλλαγμένο βάρος για κάθε ακμή που δίνεται από έναν συγκεκριμένο τύπο. Και εκεί ήταν το σημείο όπου τροποποιήσαμε τον δοθέν γράφο. (Λογικό γιατί αν έχει κάποια ακμή αρνητικά βάρη δεν θα δούλευε).

Με αφορμή τα παραπάνω λοιπόν, δημιουργήσαμε μια νέα συνάρτηση με παραλλαγή του ήδη υλοποιημένου αλγόριθμου Dijkstra, ώστε να αντιμετωπίζει το γενικευμένο πρόβλημα το συντομότερων μονοπατιών (***generalized shortest path problem***), δηλαδή, το κόστος μια διαδρομής σε έναν γράφο δεν ισούται απαραίτητα με το άθροισμα των ακμών του μονοπατιού, αλλά μπορεί να προέρχεται από μια άλλη σχέση όπως στην περίπτωση του Johnson.

Με την δημιουργία της συγκεκριμένης συνάρτησης αφαιρέθηκε ο duplicate κώδικας που βρισκόταν σε κάθε συνάρτηση για διαφορετικό output του αλγόριθμου.

Για παράδειγμα, υπήρχαν συναρτήσεις που επέστρεφαν τα μήκη των συντομότερων μονοπατιών, τα ίδια τα μονοπάτια, τις γνωστές λίστες pred, dist χρησιμοποιώντας το Dijkstra και για κάθε συνάρτηση ήταν γραμμένος ο κώδικας του relaxation loop του αλγορίθμου με πολύ μικρές αλλαγές. Συνεπώς, ήταν καλή ευκαιρία να αφαιρεθεί όλος αυτός ο διπλός κώδικας, και έτσι κάθε συνάρτηση να καλεί την δική μας συνάρτηση.

Η νέα η συνάρτηση έχει το παρακάτω σχήμα. Αξίζει να δείτε το τρίτο όρισμα *get_weight,* όπου πρόκειται μια συνάρτηση η οποία θα ορίζει το πώς θα προκύπτει το κόστος για κάθε κόμβο ώστε με βάση αυτό να υπολογιστούν τα συντομότερα μονοπάτια.

```
def _dijkstra(G, source, get_weight, pred=None,
paths=None, cutoff=None, target=None):
```

Ανοίξαμε ένα νέο pull request για το παραπάνω σκόπο, το οποίο και έγινε αποδεκτό από τα μέλη της ομάδας.

Μπορείτε να το δείτε το παρακάτω σύνδεσμο:

https://github.com/networkx/networkx/pull/1512

Προσθήκη συναρτήσεων που ελέγχουν αν ένας γράφος έχει ακμές με βάρη και ακμές με αρνητικά βάρη.

Αυτή πρόκειται για την πρώτη συνεισφορά μας και αφορά την προσθήκη δύο μεθόδων. Η μία ελέγχει αν ένας γράφος έχει ακμές με βάρη και επιστέφει True, False

ανάλογα και η δεύτερη ελέγχει αν ένας γράφος έχει ακμές με αρνητικά βάρη και επιστρέφει True, False επίσης.

Για τις συγκεκριμένες νέες συναρτήσεις προσθέσαμε ασφαλώς τις αντίστοιχες test methods, και ανοίξαμε ένα νέο pull request όπου και έγινε merge μετά από μία εβδόμαδα.

Μπορείτε να το δείτε στο παρακάτω link:

https://github.com/networkx/networkx/pull/1476

Προσθήκη προσεγγιστικών αλγορίθμων για το πρόβλημα του περιοδεύοντος πωλητή.

Το πρόβλημα του περιοδεύοντος πωλητή (TSP) είναι από τα πιο κλασικά προβλήματα συνδυαστικής βελτιστοποίησης και από τα πιο ενδιαφέροντα. Το χαρακτηριστικό αυτό του προβλήματος είναι ότι είναι NP-complete, δηλαδή δεν έχει βρεθεί πολυωνυμικός αλγόριθμος που να το λύνει. Παρόλα αυτά έχουν δημοσιευτεί κατά καιρούς αρκετοί αλγόριθμοι τεχνητής νοημοσύνης που σαν στόχο τους έχουν να λύσουν / προσεγγίσουν την βέλτιστη λύση του προβλήματος αυτού.
Τα δεδομένα του TSP μπορούν να αναπαρασταθούν σαν ένα γράφο επομένως η λύση του έχει σημασία και σε όρους γράφου, αφού πρόκειται για το Χαμιλτόνιο κύκλο (κύκλος που περνάει από όλους τους κόμβους το γράφου ακριβώς μία φορά, εκτός από το κόμβο αρχής) με το μικρότερο κόστος.

Με βάση τα παραπάνω, αποφασίσαμε να υλοποιήσουμε μερικούς τέτοιους προσεγγιστικούς αλγορίθμους για το πρόβλημα αυτό. Συγκεκριμένα έχουμε υλοποιήσει έναν αλγόριθμο προσομοιωμένης ανόπτησης (Simulated Annealing) και έναν αλγόριθμο αποδοχής κατωφλίου (Threshold Accepting).
Το NetworkX δεν έχει υλοποιήσει κάτι τέτοιο μέχρι τώρα όποτε πρόκειται για κάτι εντελώς καινούργιο του οποίου η ενσωμάτωση θα εξεταστεί σε νέες εκδόσεις του NetworkX.
Ανοίξαμε δύο pull requests που αφορούν την παραπάνω υλοποίηση. Ένα για κάθε αλγόριθμο. Τα συγκεκριμένα pull requests έχουν εξάρτηση μεταξύ τους, και συγκεκριμένα το #1554 εξαρτάται από το #1508. Τα pull requests είναι ακόμα ανοιχτά, αλλά έχουμε λάβει κάποιο feedback.

Μπορείτε να τα δείτε στους παρακάτω συνδέσμους:

#1554
https://github.com/networkx/networkx/pull/1554
#1508
https://github.com/networkx/networkx/pull/1508

Παρακάτω φαίνεται το πώς χρησιμοποιείται μία από τις συναρτήσεις που δημιουργήσαμε και συγκεκριμένα αυτή που αφορά τον αλγόριθμο Threshold

Accepting «**threshold_accepting_tsp**». Βλέπουμε ότι επιστρέφει και τον κύκλο μαζί με το κόστος του.

```
Examples
--------
>>> import networkx as nx
>>> G = nx.DiGraph()
>>> G.add_weighted_edges_from({('A', 'B', 3),
...                            ('A', 'C', 17), ('A', 'D', 14), ('B', 'A', 3),
...                           ('B', 'C', 12), ('B', 'D', 16), ('C', 'A', 13),
...                            ('C', 'B', 12), ('C', 'D', 4), ('D', 'A', 14),
...                            ('D', 'B', 15), ('D', 'C', 2)})
>>> cycle, weight = nx.threshold_accepting_tsp(G, 'D')
>>> cycle
['D', 'C', 'B', 'A', 'D']
>>> weight
31
>>> cycle, weight = nx.threshold_accepting_tsp(G, 'D',
... cycle=['D', 'B', 'A', 'C', 'D'])
>>> cycle
['D', 'C', 'B', 'A', 'D']
>>> weight
31
```

Προσθήκη αλγορίθμων για την εύρεση κοινωνιών σε ένα γράφο.

Μία άλλη κατηγορία αλγορίθμων που το NetworkX δεν υλοποιούσε είναι αυτή την εύρεσης κοινωνιών σε ένα γράφο. Συνεπώς, αποφασίσαμε να υλοποιήσουμε μερικούς από τους πολλούς αλγόριθμους που έχουν δημοσιευτεί κατά καιρούς. Συγκεκριμένα, υλοποιήσαμε δύο: τον αλγόριθμο Girvan-Newman και τον αλγόριθμο Louvain.

Για το πρώτο αλγόριθμο μετά την υλοποίηση του ανοίξαμε pull request όπου είναι ακόμα ανοιχτό και έχουμε λάβει μερικά σχόλια βελτίωσης του κώδικα.

Το σύνδεσμο για το pull request θα το δείτε παρακάτω:

https://github.com/networkx/networkx/pull/1522

Στην συνέχεια φαίνεται παράδειγμα χρήσης της συνάρτησης που δημιουργήσαμε με το όνομα «**girvan_newman**». Με το index 0 που ορίζουμε στο αποτέλεσμα παίρνουμε το πρώτο επίπεδο κοινωνίας, όπου εκεί υπάρχουν λιγότερες σε αριθμό κοινωνίες με μεγαλύτερο αριθμό κόμβων η καθεμία.

```
>>> G = nx.path_graph(10)
>>> comp = girvan_newman(G)
>>> print(comp[0])
([0, 1, 2, 3, 4], [8, 9, 5, 6, 7])
```

Όσον αφορά τον αλγόριθμο του Louvain έχουμε υλοποιήσει σχεδόν όλο το αλγόριθμο όμως θέλουμε να βελτιώσουμε ακόμα μερικά πράγματα πριν ανοίξουμε το pull request, κάτι που δεν προλάβαμε στα πλαίσια του μαθήματος αλλά θα το κάνουμε στην συνέχεια. Παρόλα αυτά μπορείτε να δείτε τα μέχρι τώρα commits που έχουν γίνει για την συγκεκριμένη συνεισφορά στο παρακάτω σύνδεσμο:

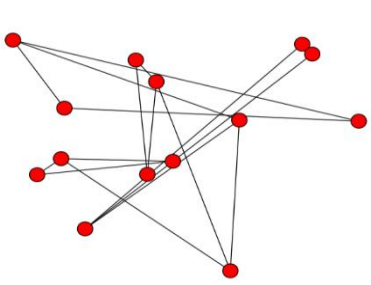
https://github.com/Overriders/networkx/tree/louvain-algorithm

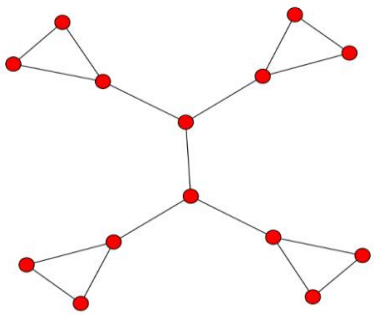
Προσθήκη αλγορίθμου για την διάταξη των κόμβων

Η συγκεκριμένη συνεισφορά, μαζί με τον αλγόριθμο του Louvain για την εύρεση των κοινωνιών μας δυσκόλεψαν πιο πολύ από όλες. Πρόκειται για την υλοποίηση ενός force-directed drawing αλγορίθμου, του Kamada-Kawai, όπως λέγεται που αφορά την διάταξη των κόμβων με τέτοιο τρόπο ώστε όταν ο γράφος οπτικοποιηθεί από κάποιο πρόγραμμα οπτικοποίησης η εικόνα που θα δούμε να είναι αισθητικά όμορφη.

Η δομή του συγκεκριμένου αλγόριθμου ήταν με μια πρώτη ματιά εύκολη στην κατανόηση της. Παρόλα αυτά στο συγκεκριμένο παράδειγμα καταλάβαμε το πόσο μακριά απέχει η αναπαράσταση ενός αλγορίθμου με ψευδοκώδικα από την υλοποίηση του σε κάποια γλώσσα προγραμματισμού. Αυτό γιατί θα πρέπει να χειριστείς την γλώσσα στην οποία διάλεξες να υλοποιήσεις την αλγόριθμο με λεπτό τρόπο, πχ να χρησιμοποιείς τις σωστές δομές δεδομένων ώστε να μην επιβαρύνεις την συνολική πολυπλοκότητα του αλγορίθμου σου, λόγω του μεγάλου όγκου των πράξεων που απαιτεί.
Οι πρώτες προσπάθειες που κάναμε οδήγησαν τον αλγόριθμο τερματίσει μετά από τρία λεπτά για έναν γράφο μόλις 50 κόμβων! Προφανώς, κάτι τέτοιο δεν είναι αλγόριθμος, συνεπώς έπρεπε να μειώσουμε σημαντικά το χρόνο. Αυτό το κάναμε χρησιμοποιώντας ένα πακέτο της Python το numpy όπου μειώσαμε το χρόνο στα 30'', στην συνέχεια στα 5'' και έπειτα από περαιτέρω βελτιώσεις στο 1'' για ένα γράφο με 50 κόμβους. Πιστεύουμε με περαιτέρω αξιοποίηση του numpy μπορούμε να πετύχουμε ακόμα καλύτερα αποτέλεσμα.

Η συνάρτηση που δημιουργήσαμε ένα λεξικό με κλειδί κάθε κόμβου του γράφου, και τιμή κάθε κλειδιού την θέση του στον Ευκλείδειο χώρο. Αν αυτό το λεξικό που επιστρέφει η συνάρτηση μας το περάσουμε σαν όρισμα σε μία από τις συναρτήσεις του NetworkX που οπτικοποιούν ένα γράφο, θα πάρουμε το παρακάτω αποτέλεσμα στα δεξιά που είναι φυσικά ανώτερο από τον ίδιο γράφο με τυχαία διάταξη των κόμβων.





Για την παραπάνω συνεισφορά ανοίξαμε ένα νέο pull request το οποίον είναι ανοιχτό μέχρι σήμερα και έχουμε λάβει feedback.

Για περισσότερες πληροφορίες, δείτε:

https://github.com/networkx/networkx/pull/1551/

3. Ποιότητα υλοποίησης

Η υλοποίηση, στο σύνολό της ακολουθεί τους κανόνες και τα πρότυπα κώδικα που ορίζει τόσο η Python, όσο και το NetworkX.
Πιο συγκεκριμένα ο κώδικας είναι στοισχισμένος (που ούτως ή άλλως το επιβάλει η Python), το μήκος των γραμμών δεν ξεπερνάει τους 80 χαρακτήρες, η τεκμηρίωση του κώδικα είναι πολύ αναλυτική (και σύμφωνη με τα πρότυπα του NetworkX) και γενικότερα έχει γίνει προσπάθεια, ώστε, όπου είναι δυνατόν να ακολουθείται ο “pythonic” τρόπος γραψίματος κώδικα (όπως είναι επιθυμητό και από την κοινότητα του NetworkX).

4. Ολοκλήρωση

Η συνεισφορά μας δεν επιδρά αρνητικά στον υπόλοιπο κώδικα και είναι χωρίς λάθη, δηλαδή τρέχει έτσι όπως αναμέναμε κάτι που το διασφαλίζει το Travis CI που χρησιμοποιείται στο NetworkX. Επομένως, κάθε pull request που ανοίξαμε πυροδοτούσε εργασία του Travis που έλεγχε αν όλα τα tests που υπάρχουν στο έργο περνάνε με επιτυχία σε όλες τις εκδόσεις της Python, από την 2.7 μέχρι την 3.4. Επίσης, σε συνδυασμό με το coveralls ελέγχεται κατά πόσο ο κώδικας μας επηρεάζει το συνολικό ποσοστό κάλυψης του κώδικα. Αν ο κώδικας μας μειώνει το συνολικό ποσοστό κάλυψης κατά 1% τότε δεν περνάει τον έλεγχο από το coveralls, συνεπώς δεν γίνεται να υπάρξει συνεισφορά στο NetworkX δεκτή, αν δεν υλοποιηθούν τα απαραίτητα test για αυτήν.

5. Έλεγχος

Ο κώδικας που έχει γραφεί συνοδεύεται από τον αντίστοιχο έλεγχο μονάδων. Δεν γινόταν ούτως η άλλως να γίνει συνεισφορά δεκτή χωρίς τα αντίστοιχα tests. Στην προκειμένη περίπτωση, ο έλεγχος μονάδων πραγματοποιείται με nosetests, όπως ορίζει το NetworkX. Η κάλυψη μάλιστα του κώδικα των συναρτήσεων που υλοποιήσαμε είναι 100%.

6. Συνεργασία με την ομάδα ανάπτυξης

Η συνεργασία μας με την ομάδα ανάπτυξης ήταν πολύ καλή. Στην αρχή, επικοινωνήσαμε μέσω e-mail με τους authors του NetworkX και τους είπαμε για τη συνεισφορά που σκεφτόμαστε να κάνουμε στο έργο. Αυτοί, μας απάντησαν φιλικά να προχωρήσουμε στη συνεισφορά και να ανοίξουμε σχετικό pull request ή issue στο GitHub. Στη συνέχεια, για κάθε pull request που ανοίξαμε πήραμε το απαραίτητο feedback και συζητήσαμε πάνω στον κώδικα που είχαμε γράψει, από μέλη της κοινότητας του NetworkX, ώστε να κάνουμε τις απαραίτητες βελτιώσεις. Αφού κάναμε τις απαραίτητες αλλαγές (για τα πρώτα 3 pull requests), τα ενσωμάτωσε στον κώδικα του έργου (στα υπόλοιπα ακόμα γίνονται σχόλια).

7. Παρουσιάσεις

Οι παρουσιάσεις πήγαν πολύ καλά, παρά τον ίσως περιορισμένο χρόνο που είχαμε στη διάθεσή μας (για την παρουσίαση). Μετά το πέρας της κάθε παρουσίασης μας έγιναν ερωτήσεις σχετικά με αυτήν, κάτι το οποίο δείχνει ότι υπήρχε ενδιαφέρον από τους συμφοιτητές μας, κάτι το οποίο φάνηκε και από τις βαθμολογίες τους, αφού τις 2 πρώτες παρουσιάσεις μας τις κατέταξαν ανάμεσα στις 3 καλύτερες.

8. Οργάνωση στο GitHub (wiki, issues, commits, branching)

Για να δουλέψουμε στο συγκεκριμένο έργο, δημιουργήσαμε έναν νέο οργανισμό στο Github με όνομα Overriders όπου μέλη του οργανισμού είναι η ομάδα μας. Στον οργανισμό Overriders δημιουργήσαμε το forked αποθετήριο που αφορούσε το NetworkX. Το αποθετήριο μας βρίσκεται στο παρακάτω σύνδεσμο: https://github.com/Overriders/networkx/

Στην συνέχεια, για κάθε διαφορετική υλοποίηση αλγορίθμου (ή γενικά συνεισφοράς) δημιουργήσαμε ξεχωριστό κλάδο όπου εκεί κάναμε τις αλλαγές. Για παράδειγμα, για την υλοποίηση του αλγόριθμου Johnson φτιάξαμε ένα branch με όνομα *johnson,* για την υλοποίηση του αλγόριθμου Kamada-Kawai φτιάξαμε ένα branch με όνομα *kamada-kawai,* κ.ο.κ. Συνολικά, δημιουργήσαμε 9 νέους κλάδους. Κατά την διάρκεια των αλλαγών, κάθε μέλος έκανε review σε πολλά από τα commits που έκανε το άλλο μέλος σχολιάζοντας τις αλλαγές που μόλις έγιναν, όπως για παράδειγμα φαίνεται στο παρακάτω commit:

https://github.com/Overriders/networkx/commit/8a9168f4c478824f2db9de9900a654b66f093f15

Κάθε φορά που τελειώναμε μία συγκεκριμένη συνεισφορά ανοίγαμε ένα νέο pull request. Επίσης, χρησιμοποιήσαμε πολλές εντολές του git, με πιο σημαντικές *git*

pull, git merge, git clone, git push, git commit, git rebase, git fetch, όπου ήταν απαραίτητες για την διαχείριση του έργου μας.

Για πληροφορίες για τα πιο σημαντικά γεγονότα που έγιναν κατά την διάρκεια της συνεισφορά μας μπορείτε να δείτε το ιστολόγιό μας στη παρακάτω διεύθυνση.

http://overriders.blogspot.com/