Εργασία Θεωρίας Δικτύων

Φίλιππος Ρωσσίδης
(ΑΕΜ 10379)

5 Ιανουαρίου 2025

1 Εισαγωγή

Στην εργασία αυτή εξετάζουμε την αναγνώριση κοινωτήτων. Συγκεκριμένα θα υλοποιούμε την distance quality function θα ελέγξουμε κατά πόσο αυτή είναι καλή επιλογή.

Οι συναρτήσεις αυτές (quality functions) δέχονται ως όρισμα έναν γράφο και μια κατανομή κοινοτήτων και επιστρέφουν μία τιμή. Σκοπώς μας είναι να επιστρέφουν υψηλές τιμές για κατανομές που παρουσιάζουν ισχυρή κοινοτική συμπεριφορά, ώστε να μπορούμε μεγιστοποιώντας αυτή τη συνάρτηση να ανιχνεύσουμε τις κοινότητες.

Για τον έλεγχο της ποιότητας χρησιμοποιούνται τεχνικές όπως δημιουργία γράφων με πολύ ισχυρές κοινότητες και σταδική τυχαία μετάλαξη τους, εξέταση σε πραγματικούς γράφους των οποίων τις κοινότητες ήδη γνωρίζουμε, και επίσης σύγκριση με την modularity.

2 Η συνάρτηση distance quality

2.1 Μαθηματικός Ορισμός

Η συνάρτηση δέχεται σαν όρισμα έναν γράφο και κάποια κατανομή κοινοτήτων.

Ορίζουμε:

$$D_V(i,j) = \min\{k : A_G^k(i,j) \neq 0\}$$

τον πίνακα που περιέχει τις αποστάσεις μεταξύ κάθε δύο κόμβων του γράφου i, j, όπου $A_G^k(i,j)$ ο πίνακας γειτνίασης του γράφου G. υψομένος στην k δύναμη.

Ορίζουμε επίσης:

$$D_V(C) = \sum_{i,j \in C} D_V(i,j)$$

Αυτό αποτελεί μέτρο του πόσο στενά συνδεδεμένες είναι μεταξύ τους οι κοινότητες. Επιθυμούμε να το συγκρίνουμε με την αναμενώμενη τιμή για τυχαίο γράφο.

Για να ορίσουμε τον τυχαίο γράφο επιλέγουμε το γενικευμένο μοντέλο $(\cdots)$, όπου συνδέουμε τυχαία ακμές μεταξύ κόμβων, διατηρώντας όμως τον βαθμό τους.

Αν $d_k(v)$ είναι ο k βαθμός του κόμβου v, δηλαδή το πλήθος των ελάχιστων μονοματιών μήκους k που ξεκινάν από τον v και $m_k(G) = \frac{1}{2}\sum_{v \in V(G)} d_k(v)$ το πλήθος ελάχιστων μονοπατιών μήκους k στον γράφο G, τότε η πιθανότητα δύο κόμβοι $i, j \in G$ να συνδέονται με ακμή μήκους k είναι:

$$Pr[i,j,k] = \frac{d_k(i)}{2m_k(G)} \frac{d_k(j)}{2m_k(G)}$$

η αναμενώμενη απόσταση των κόμβων:

$$\overline{D_V(i,j)} = \sum_{k=1}^{diam(G)} kPr[i,j,k]$$

όπου $diam(G)$ η διάμετρος του γράφου (μέγιστο ελάχιστο μονοπάτι).

Αν ορίσουμε

$$\overline{D_V(C)} = \frac{1}{2}\sum_{i,j\in C} \overline{D_V(i,j)}$$

το άθροισμα των αναμενώμενων αποστάσεων των κόμβων στην κοινότητα C, τότε η συνάρτηση:

$$Q_d(G,C) = \sum_{c\in C}(\overline{D_V(C)} - D_V(C))$$

δηλώνει την επιθυμητή σύγκριση του μέτρου στενής σύγκρισης των κοινοτήτων C ως προς του τυχαίου γράφου.

Περιμένουμε οι τιμές της Q_d να είναι υψηλές για ισχυρές δομές κοινοτήτων.

Στην υλοποίηση μου διάλεξα επίσης να τροποποιήσω την συνάρτηση ως εξίς:

$$D_V(C) = \sum_{i,j\in C} D_V(i,j)$$

$$\overline{D_V(C)} = \sum_{i,j\in C} \overline{D_V(i,j)}$$

$$Q_d(G,C) = \sum_{c\in C}[(1-\gamma)\overline{D_V(C)} - \gamma D_V(C)]$$

ώστε για $\gamma = 0.5$ να ισοδυναμεί με τον προϊγούμενο ορισμό, αλλά να έχω δυνατότητα επιλογής ποιός όρος θα επηρρεάσει περισσότερο.

2.2 Υλοποίηση

Χρησιμοποιήθηκε δυναμική προσέγγιση, δηλαδή οι τιμές του αλγορίθμου που χρησιμοποιούνται συχνά ($D_V(i,j)$, $d_k(i)$, $m_G(k)$, $Pr[i,j,k]$, $\overline{D_V(i,j)}$, ..) υπολογίζονται μία φορά στην αρχή και αποθηκεύονται. Έτσι γλυτώνουμε τις άσκοπες επαναλύψεις. Ειδικότερα στην περίπτωση του συγκεκριμένου αλγορίθμου, αφού η συνάρτηση ποιότητας θα υπολογιστεί χιλιάδες φορές για διαφορετικές επιλογές κοινοτήτων στην διαδικασία μεγιστοποίησής της, και εφόσον οι μακράν πιο αργές διαδικασίες (όπως θα περιγραφτεί παρακάτω) είναι αυτές των προαναφερθέντων ποσοτήτων, η διαφορά στο χρόνο εκτέλεσης είναι ραγδαία.

Όπου είναι δυνατό χρησιμοποιούνται συναρτήσεις της βιβλιοθήκης $networkx$ μιας και είναι βελτιστοποιημένες και θα συμβάλουν στην ταχύτητα του κώδικα.

Στην πρώτη φάση του αλγορίθμου υπολογίζονται οι παρακάτω ποσότητες και αποθηκεύονται για μελλοντική χρήση:

- **διάμετρος του γράφου**

  Χρησιμοποιείται η συνάρτηση της βιβλιοθήκης $networkx$: $diameter(G)$.

- $\mathbf{D_V(i,j)}$

  Χρησιμοποιείται η συνάρτηση της βιβλιοθήκης $networkx$: $all\_pairs\_shortest\_path\_length(G)$, η οποία επιστρέφει απευθείας το ζητούμενο, την απόσταση κάθε δύο κόμβων του γράφου μεταξύ τους. Τα στοιχεία αποθηκεύονται σε ένα *python dict*. Η χρονική πολυπλοκότητα της συνάρτησης αυτής είναι $O(|V|^2 \log|V| + |V||E|)$ (;), όπου $|V|$ το πλήθος κόμβων του γράφου και $|E|$ το πλήθος των ακμών.

- $\mathbf{d_k(i)}$ και $\mathbf{m_G(k)}$

  Χρησιμοποιείται η συνάρτηση της βιβλιοθήκης $networkx$: $singe\_source\_shortest\_path\_length(G, source)$, η οποία υπολογίζει για κάποιον κόμβο $source$ όλα τα ελάχιστα μονοπάτια που τον έχουν σαν άκρο. Αν υπολογιστούν αυτά τα μονοπάτια για κάθε κόμβο ως $source$ τότε θα έχουμε υπολογίσει όλα τα ελάχιστα μονοπάτια του γράφου δύο φορές (μία για $i \rightsquigarrow j$ και μια για $j \rightsquigarrow i$). Η συνάρτηση χρησιμοποιεί την τεχική $Breadth\ First\ Search$, έτσιη χρονική πολυπλοκότητα της είναι $O(|V| + |E|)$

  Τρέχουμε την παραπάνω συνάρτηση για κάθε κόμβο του γράφου ($source$). Στο $dict$ που αυτή επιστρέφει, περιέχονται για κάθε άλλο κόμβο ($target$) μια λίστα με μήκη από όλα τα μονοπάτια που τους ενώνουν ($source \rightsquigarrow target$). Έτσι για κάθε μονοπάτι που βρίσκουμε στη λίστα, έχουμε το μήκος του k, αυξάνουμε κατά 1 τον βαθμό $d_k(source)$ και κατά 0.5 τη τιμή $m_G(k)$. Χρησιμοποιούμε 0.5 διότι $m_k(G) = \frac{1}{2}\sum_{v \in V(G)} d_k(v)$ (μετράμε κάθε μονοπάτι 2 φορές).

  Η πολυπλοκότητα της παραπάνω διαδικασίας είναι: $\#addcomplexity$.

- $\mathbf{Pr[i, j, k]}$ και $\overline{\mathbf{D_V(i, j)}}$

  Υπολογίζοντα απλά με τους τύπους:

$$Pr[i, j, k] = \frac{d_k(i)}{2m_k(G)} \frac{d_k(j)}{2m_k(G)}$$

$$\overline{D_V(i, j)} = \sum_{k=1}^{diam(G)} kPr[i, j, k]$$

  και αποθηκεύονται.

  Συγκεκριμένα οι πίνακες αυτοί είναι συμμετρικοί (αφού $\frac{d_k(i)}{2m_k(G)} \frac{d_k(j)}{2m_k(G)} = \frac{d_k(j)}{2m_k(G)} \frac{d_k(i)}{2m_k(G)}$) οπότε οι τιμές αυτές υπολογίζονται μία φορά για κάθε ζευγάρι i, j και προστίθονται στο τέλος μία φορά για $i = j$ και δύο για $i \neq j$.

  Η χρονική πολυπλοκότητα της παραπάνω διαδικασίας είναι: $\#addcomplexity$.

Τα παραπάνω υπολογίζονται μία φορά στην αρχή του αλγορίθμου και αποθηκεύονται για μελλοντική χρήση, ώστε κάθε φορά που χρειάζονται (θα χρειαστούν πολλαπλές φορές το καθένα στο στάδιο της μεγιστοποίησης του Q_d) η χρονική πολυπλοκότητα κλήσης τους είναι $O(1)$.

Επιπλέον προσοχή χρειάζεται το γεγονός ότι οι παραπάνω διαδικασίες δε μπορούν να λειτουργήσουν για μη συνδεδεμένους γράφους. Έτσι υιοθετούμε τη σύμβαση ότι δύο μη συνδεδεμές συνεκτικές συνιστώσες δεν μπορούν να βρίσκονται στην ίδια κοινώτητα. Όποτε διαχωρίζουμε τον γράφο στις συνεκτικές του συνιστώσες και τρέχουμε τα παραπάνω για κάθε μία ξεχωριστά. Αυτή η προσέγγιση διατηρεί την πολυπλοκότητα ίδια (αφού οι συνεκτικές συνιστώσες διαμερίζουν τον γράφο).

Συνολική πολυπλοκότητα της πρώτης φάσης: $addcomplexity$.

Στη δεύτερη φάση του αλγορίθμου, αφού έχουν αποθηκευτεί τα παραπάνω, υπολογίζονται για τις δοσμένες κοινότητες C τα αθροίσματα:

$$D_V(C) = \sum_{i,j \in C} D_V(i, j), \qquad \overline{D_V(C)} = \sum_{i,j \in C} \overline{D_V(i, j)}$$

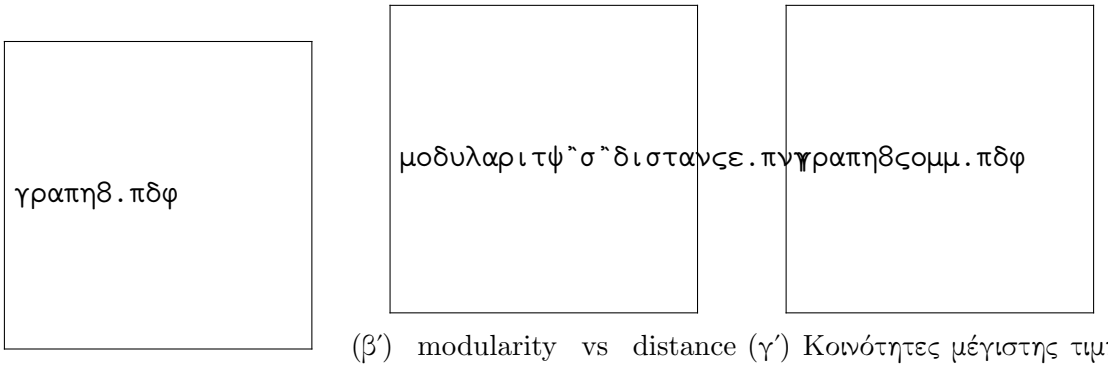
για $i, j \in C$, και η τιμή:

$$Q_d(G, C) = \sum_{c \in C} [(1 - \gamma)\overline{D_V(C)} - \gamma D_V(C)]$$

Αυτό είναι το κομμάτι που θα επαναλαμβάνεται για τις δοκιμές των κατανομών κοινοτήτων στο κομμάτι της βελτιστοποίησης.

Συνολική πολυπλοκότητα της δεύτερης φάσης: $O(|V|^2)$.

2.2.1 Μια πρώτη ματία στα αποτελέσματα



(α′) Γράφος

(β′) modularity vs distance quality

(γ′) Κοινότητες μέγιστης τιμής Q_d

Σχήμα 1: Ώμη Δύναμη

Δοκιμάστηκε για το γράφο του Σχήματος ;; τεχική ωμής δύναμης. Στο σχήμα ;; φαίνονται στον άξονα x οι τιμές της *modularity* και στον άξονα y οι τιμές της *distance quality function* για κάθε πιθανό συνδιασμό κοινοτήτων.

2.2.2 Μεγιστοποίηση του Q_d