ΜΑΥΡΟΦΟΡΑΚΗ ΔΗΜΗΤΡΑ
1115201400104

1η ΕΡΓΑΣΙΑ

Υλοποίηση δομής για την εύρεση κοντινών γειτόνων στη γλώσσα C/C++

Το πρόγραμμά μου υλοποιεί τον αλγόριθμο LSH με μετρική το συνημίτονο και την ευκλείδια απόσταση, καθώς και την μέθοδο τυχαίας προβολής σε υπερκύβο χρησιμοποιώντας ξανά τις παραπάνω μετρικές. Συνεπώς για κάθε σημείο ενός δοθέντος queryset οι αλγόριθμοι βρίσκουν τον κοντινότερο προσεγγιστικά αλλά και εξαντλητικά γείτονα κάθε vector και ταυτόχρονα υπολογίζουν τους χρόνους εύρεσής τους.

- **Διάρθρωση αρχείων κώδικα**

lsh.cc : Η main function του αλγορίθμου lsh. Σ’ αυτό το αρχείο διαβάζω το dataset, το αποθηκεύω σε έναν πίνακα και στην συνέχεια δημιουργώ τους hash tables ανάλογα με το L και εισάγω ένα ένα τα σημεία του dataset. Στη συνέχεια διαβάζω το queryset και για κάθε σημείο εκτελώ τους αλγορίθμους rangesearch (εύρεση των κοντινών γειτόνων βάσει μετρικών) , approximateNN (ευρεση του πιο κοντινού γείτονα βάσει μετρικών) , trueNN (εύρεση του κοντινού γείτονα βάσει εξαντλητικής αναζήτησης). Τέλος εκτυπώνω τα αποτελέσματα στο output file.

lsh_funcs.h : Header των αρχείων που αφορούν μόνο τον lsh. Σε αυτή τη φάση περιλαμβάνει μόνο τους αλγορίθμους εύρεσης γειτόνων, αλλά για λόγους επεκτασιμότητας και ανεξαρτησίας των αρχείων, μπορεί να χρειαστεί να προστεθούν και άλλες συναρτήσεις.

lsh_funcs.cc : Υλοποίηση των συναρτήσεων που αφορούν μόνο τον lsh.

cube.cc : Η main function του αλγορίθμου προβολής σε υπερκύβο. Είναι διαρθρωμένο με πολύ παρόμοιο τρόπο με τον lsh.cc με τη διαφορά ότι πλέον το dataset αντί για hashtables αποθηκεύεται σε έναν και μοναδικό πίνακα. Οι αλγόριθμοι εύρεσης γειτόνων πραγματοποιούνται με όρισμα αυτόν τον πίνακα.

cube_funcs.h : Header των αρχείων που αφορούν μόνο τον υπερκύβο. Περιλαμβάνει κυρίως τους αλγορίθμους εύρεσης γειτόνων (παραμετροποιημένους για τη δομή του υπερκύβου).

cube_funcs.cc : Υλοποίηση των συναρτήσεων που αφορούν μόνο τον υπερκύβο.

datastructs.h : Header που περιλαμβάνει όλες τις κοινές δομές του υπερκύβου και του lsh. Συγκεκριμένα, τις κλάσεις που έχω δημιουργήσει , τις συναρτήσεις που υπολογίζουν τις

αποστάσεις, τις συναρτήσεις που υπολογίζουν τις παραμέτρους t, r, v που χρησιμοποιούνται για την εύρεση του Key κ.α.

datastructs.cc : Υλοποιεί τις συναρτήσεις του datastructs.h

Makefile: Αρχείο για την μεταγλώττιση του προγράμματος.

- **Σχόλια για την επιλογή των δομών**

Η βασική κλάση των σημείων που διαβάζονται από τα αρχεία είναι η DataVector. Ανάλογα με το αν έχει επιλεγεί να χρησιμοποιηθεί ευκλείδια μετρική ή μετρική συνημιτόνου δημιουργούνται δυναμικά objects τυπου Cosine ή Euclidean (κλάσεις παιδιά της DataVector). Ωστόσο όλες οι συναρτήσεις έχουν όρισμα τύπου DataVector * για να είναι generic.

Τα v, t, r είναι διδιαστατοι πινακες και δημιουργούνται μία φορά στην αρχή του προγράμματος. Οι v, t έχουν διάσταση L*k ενώ ο r είναι διάστασης 1 * k ωστόσο τον αντιμετωπίζω σαν διδιάστατο για να εκμεταλλευθώ τις συναρτήσεις που έχω ήδη φτιάξει.

Για την υλοποίηση του hashtable έχω χρησιμοποιήσει την έτοιμη δομη unordered_map της stl και την hash function που η βιβλιοθήκη μου δίνει.

Σημείωση: Δεν έχω επέμβει στον τρόπο με τον οποίο η hash function της stl κατανέμει τα σημεία σε buckets γιατί επιλέγει να δημιουργήσει τόσα buckets όσα είναι απαραίτητα για να αποφευχθούν τα collisions.

Το euclidean key με το οποίο εισάγεται ένα σημείο στους hashtables είναι το διάνυσμα g σε string.
Το cosine key είναι και αυτό ένα bitstring.

Ο υπερκύβος είναι ένας πίνακας από λίστες με σημεία. Για να εισαχθεί ένα σημείο στον πίνακα, έχω κατλήξει σε ένα bitstring το οποίο το μετατρέπω στο αντίστοιχο δεκαδικό νούμερο. Αυτό το νούμερο δηλώνει το κελί στο οποίο τελικά θα εισαχθεί το σημείο.

Σημείωση : Το d' (διάσταση του πίνακα) προκύπτει αποκλειστικά από το input που θα δώσει ο χρήστης και έχει default = 3.

- **Μεταγλώττιση**

make -> δημιουργεί εκτελέσιμα αρχεία και για τον υπερκύβο και για τον lsh
make cube -> δημιουργεί εκτελέσιμα αρχείο μόνο για τον υπερκύβο
make lsh -> δημιουργεί εκτελέσιμο αρχείο μόνο για τον lsh

- **Εκτέλεση**

./lsh -d sample_datasets/Ergasia_1/siftsmall/input_small -q sample_da
tasets/Ergasia_1/siftsmall/query_small -o output_file

./cube -d sample_datasets/Ergasia_1/siftsmall/input_small -q sample_da
tasets/Ergasia_1/siftsmall/query_small -o output_file -M 2000

Έχω ως πρότυπο αρχείων για input , την μορφοποίηση που δόθηκε στο eclass.
Για να τερματίσει το πρόγραμμα θα πρέπει να δοθεί από τη γραμμή εντολών “no” στην ερώτηση αν θέλει να συνεχίσει.

- **Παρατηρήσεις αλγορίθμων (small dataset)**

LSH euclidean :

Ενδεικτικές εκτελέσεις :

- *k=4 , L=5*

Max fraction: 293.625612/141.463776 = 2.07562
Mean time tLSH 17.7651

- *k= 8, L=5* (k>default)

Max fraction: 399.703640/99.629313 = 4.01191
Mean time tLSH 0.13831
Για μεγαλύτερο αριθμό hash function , το κλειδί που δημιουργείται έχει μεγαλύτερο μέγεθος. Συνεπώς τα στοιχεία που βρίσκονται σε κάθε bucket είναι λιγότερα (πλέον υπάρχει μεγαλύτερη εξατομίκευση). Γι αυτό το λόγο έχουμε μικρότερη ακρίβεια αλλά γρηγορότερη εύρεση του κοντινότερου γείτονα.

- *k= 8, L=5* (L > default)

Max fraction: 293.625612/141.463776 = 2.07562
Mean time tLSH 31.3248
Για μεγαλύτερο αριθμό hashtables αυξάνεται κατα πολύ ο χρόνος εύρεσης του προσεγγιστικά κοντινότερου καθώς θα πρέπει να διατρέξουμε περισσότερες δομές. Ωστόσο η ακρίβεια δεν αλλάζει καθώς δεν έχει αλλάξει εσωτερικά η κατανομή των στοιχείων σε κάθε hash table.

Cosine euclidean

LIMITS : 1800 (τα σημεία που ελέγχεις σε κάθε bucket)
Η cosine μετρική δημιουργεί λιγότερα buckets στον πινακα γιατί πλέον είναι bitstring κωδικοποιήσεις και όχι strings απο integers. Συνεπώς υπάρχει συσσώρευση σημείων στα buckets. Επιλέγω τη συγκεκριμένη τιμή LIMITS ώστε ο approximate να έχει ισορροπία ακρίβειας και ταχύτητας.

- *k= 4, L=5*

Max fraction: 0.192677/0.105538 = 1.82567

Mean time tLSH 96.3124

Αν αυξομειώσουμε τις τιμές των k, L θα παρατηρήσουμε ίδια συμπεριφορά με την ευκλείδια μετρική.

Γενικά η μετρική του συνημιτόνου μπορεί να μας παρέχει μεγαλύτερη ακρίβεια καθώς συσσωρεύονται περισσότερα στοιχεία στο ίδιο bucket, αλλά χρονικά ειναι πιο χρονοβόρα καθώς θα πρέπει να ελέγχει περισσότερα σημεία.

Hypercube cosine/euclidean

Τα αποτελέσματα που δίνουν είναι ανάλογα καθώς και στις δύο περιπτώσεις καταλήγουμε να κατανείμουμε τα σημεία σε έναν πίνακα 2^k θέσεων. Ενδεικτικά τα αποτελέσματα του hypercube cosine:

- *k= 3, M = 2000, probes = 2*

Max fraction: 0.217475/0.049688 = 4.37682
Mean time tLSH 38.215

- *k= 10, M = 2000, probes = 2*

Max fraction: 0.047183/0.004903 = 9.62383
Mean time tLSH 5.80025
Όπως και με τον lsh αυξάνοντας το k, αυξάνουμε το μέγεθος του πίνακα. Συνεπώς έχουμε περισσότερα “κελιά” με λιγότερα στοιχεία.

- *k= 10, M = 2000, probes = 10*

Max fraction: 0.047183/0.004903 = 9.62383
Mean time tLSH 6.29163

- *k= 10, M = 100, probes = 2*

Max fraction: 0.137346/0.012590 = 10.9091
Mean time tLSH 2.08835
Ψάχνοντας λιγότερα σημεία , μειώνεται και η πιθανότητα να βρούμε τον πραγματικά πιο κοντινό γείτονα.

Ενδεικτικά τα αποτελέσματα του hypercube euclidean:

- *k= 3, M = 2000, probes = 2*

Max fraction: 323.589246/65.741920 = 4.92211
Mean time tLSH 87.2172

- **Σύγκριση cosine lsh και cosine hypercube (small dataset)**

Για τον LSH ισχύει:

➢ *k= 12, L=8*

Max fraction: 0.181073/0.038724 = 4.67596
Mean time tLSH 7.24535

Για τον hypercube ισχύει:

➢ *k= 3, M = 2000, probes = 2*

Max fraction: 0.217475/0.049688 = 4.37682
Mean time tLSH 38.215

Παρατηρούμε ότι για παρόμοιο μέγιστο κλάσμα προσέγγισης ο hypercube είναι πολύ αργότερος. Αυτό εξηγείται από το γεγονός ότι τα σημεία που θα πρέπει να ελέγξει σε κάθε bucket είναι περισσότερα.

- **Χωρική πολυπλοκότητα**

Ο υπερκύβος καταναλώνει λιγότερα bytes μνήμης από τον LSH γιατί αποτελείται από έναν πίνακα με λίστες από pointers, ενώ ο Lsh αποτελείται από έναν πίνακα με unordered_maps οι οποίοι εσωτερικά περιέχουν λίστες από pointers.

ΣΥΜΠΕΡΑΣΜΑ

Ο lsh είναι πολύ πιο γρήγορος και ακριβής τρόπος εύρεσης κοντινών γειτόνων αλλά όσον αφορά την χωρική πολυπλοκότητα είναι κοστοβόρος. Σε αντίθεση ο υπερκύβος δεν παρέχει τόσο ακριβή αποτελέσματα ωστόσο σαν δομή είναι πιο λιτή και συνεπώς λιγότερο κοστοβόρα. Ανάλογα με το τι επιθυμούμε να επιτύχουμε (καλύτερη χρονική ή χωρική πολυπλοκότητα), επιλέγουμε και την αντίστοιχη δομή.