Ανάπτυξη λογισμικού για αλγοριθμικά προβλήματα

Μέρος 2ο

Μέλη ομάδας:
Ευριπίδης Παντελαίος - sdi1600124
Κρικόρ Τζεβαχιριάν - sdi1700162

Έχουν υλοποιηθεί όλα τα ζητούμενα της εργασίας. Οι παραδοχές οι οποίες χρειάστηκαν για την υλοποίηση της εργασίας αναφέρονται παρακάτω, όπως και ενδεικτικές εκτελέσεις μαζί με τα αποτελέσματά τους.

Μεταγλώττιση των αρχείων

Η μεταγλώττιση γίνεται με την εντολή **make** και παράγονται 2 διαφορετικά εκτελέσιμα αρχεία, τα οποία είναι για το πρώτο μέρος με όνομα '**search**' και για το δεύτερο μέρος της εργασίας αντίστοιχα με όνομα '**cluster**'.

Για την χρήση των unit tests θα πρέπει να γίνει εγκατάσταση των εξής πακέτων σε linux συστήματα με την εντολή:
git clone https://github.com/mity/acutest.git

< Το header "acutest.h" που χρειάζεται για να γίνει το unit testing συμπεριλαμβάνεται στα παραδοτέα αρχεία >
Το unit testing χρησιμοποιήθηκε καθ' όλη την διάρκεια ανάπτυξης και συγγραφής του κώδικα του βασικού προγράμματος και ο έλεγχος που κάνουμε αφορά κυρίως τα μεγέθη των καμπυλών και το πλήθος των σημείων των παραγόμενων vectors πριν υπολογιστεί το κλειδί για την εισαγωγή των vectors στα hashtables. Επίσης, έχουμε έναν έλεγχο ομαλής λειτουργίας, όπου ελέγχουμε γενικότερα τις συναρτήσεις του προγράμματος ότι λειτουργούν σωστά.

Ενδεικτικές εκτελέσεις και αποτελέσματα

Μέρος Α)

Discrete Frechet:

1) ./search -i nasd_input.csv -q nasd_query.csv -k 4 -L 5 -algorithm Frechet -metric discrete -delta 1
(**100** input curves and **10** query curves *with **730** number of points for each curve)*

```
tApproximateAverage: 152.595 ms
tTrueAverage: 196.959 ms
MAF: 1.28627
[Extra] Average Approximation Factor: 1.02863
```

2) ./search -i big_input.csv -q big_query.csv -k 4 -L 5 -algorithm Frechet -metric discrete -delta 1.2
*(**6548** input curves and **30** query curves with **120** number of points for each curve)*

```
tApproximateAverage: 41.9403 ms
tTrueAverage: 409.129 ms
MAF: 1.67278
[Extra] Average Approximation Factor: 1.12072
```

3) ./search -i input_small_id -q query_small_id -k 4 -L 5 -algorithm Frechet -metric discrete -delta 1
*(**10000** input curves and **100** query curves with **128** number of points for each curve)*

```
tApproximateAverage: 11.1817 ms
tTrueAverage: 916.492 ms
MAF: 1.63897
[Extra] Average Approximation Factor: 1.13941
```

Continuous Frechet:

1) ./search -i nasd_input.csv -q nasd_query.csv -algorithm Frechet -metric continuous -delta 1
(**100** input curves and **10** query curves *with **730** number of points for each curve)*

```
tApproximateAverage: 622.573 ms
tTrueAverage: 1802.43 ms
MAF: 2.44798
[Extra] Average Approximation Factor: 1.26728
```

2) ./search -i big_input.csv -q big_query.csv -algorithm Frechet -metric continuous -delta 1.2
*(**6548** input curves and **30** query curves with **120** number of points for each curve)*

```
tApproximateAverage: 702.513 ms
tTrueAverage: 2063.53 ms
MAF: 1.85614
[Extra] Average Approximation Factor: 1.18577
```

LSH:

1) ./search -i nasd_input.csv -q nasd_query.csv -k 4 -L 5 -algorithm LSH
(**100** input curves and **10** query curves *with **730** number of points for each curve)*

```
tApproximateAverage: 0.5255 ms
tTrueAverage: 1.31697 ms
MAF: 1.43312
[Extra] Average Approximation Factor: 1.01877
```

2) ./search -i input_small_id -q query_small_id -k 4 -L 5 -algorithm LSH
*(**10000** input curves and **100** query curves with **128** number of points for each curve)*

```
tApproximateAverage: 0.73887 ms
tTrueAverage: 2.75161 ms
MAF: 1.46767
[Extra] Average Approximation Factor: 1.02769
```

Hypercube:

1) ./search -i nasd_input.csv -q nasd_query.csv -M 20 -probes 3 -algorithm Hypercube
(**100** input curves and **10** query curves *with **730** number of points for each curve)*

```
tApproximateAverage: 0.0933 ms
tTrueAverage: 0.2322 ms
MAF: 1.85178
[Extra] Average Approximation Factor: 1.47225
```

2) ./search -i input_small_id -q query_small_id -M 20 -probes 3 -algorithm Hypercube
*(**10000** input curves and **100** query curves with **128** number of points for each curve)*

```
tApproximateAverage: 0.85724 ms
tTrueAverage: 2.63974 ms
MAF: 1.79154
[Extra] Average Approximation Factor: 1.13916
```

Μέρος Β)

Για τις παραμέτρους έγιναν αρκετές δοκιμές, ενδεικτικά παρουσιάζουμε μερικές εκτελέσεις στο input.csv με αριθμό cluster = 3, το οποίο παρέχει ικανοποιητικά αποτελέσματα.

LSH Frechet Mean Curve:

1) ./cluster -i nasd_input.csv -c cluster.conf -update mean_vector -assignment LSH -silhouette
(**100** input curves *with **730** number of points for each curve)*

```
CLUSTER-1 size: 63
CLUSTER-2 size: 2
CLUSTER-3 size: 35
Silhouette: [0.549313, 0.693345, -0.236766, stotal: 0.335297]
```

Lloyd’s Mean Curve:

1) ./cluster -i nasd_input.csv -c cluster.conf -update mean_frechet -assignment Classic -silhouette
(**100** input curves *with **730** number of points for each curve)*

```
CLUSTER-1 size: 16
CLUSTER-2 size: 2
CLUSTER-3 size: 82
Silhouette: [0.270078, 0.661664, 0.701603, stotal: 0.544448]
```

Lloyd’s Mean Vector:

1)./cluster -i nasd_input.csv -c cluster.conf -update mean_vector -assignment Classic -silhouette
(**100** input curves *with **730** number of points for each curve)*

```
CLUSTER-1 size: 69
CLUSTER-2 size: 6
CLUSTER-3 size: 25
Silhouette: [0.670267, 0.330919, 0.342401, stotal: 0.447862]
```

LSH Mean Vector:

1)./cluster -i nasd_input.csv -c cluster.conf -update mean_vector -assignment LSH -silhouette
(**100** input curves *with **730** number of points for each curve)*

```
CLUSTER-1 size: 57
CLUSTER-2 size: 2
CLUSTER-3 size: 41
Silhouette: [0.438786, 0.652703, -0.125656, stotal: 0.321944]
```

Hypercube Mean Vector:

1) ./cluster -i nasd_input.csv -c cluster.conf -update mean_vector -assignment Hypercube -silhouette
(**100** input curves *with **730** number of points for each curve)*

```
CLUSTER-1 size: 67
CLUSTER-3 size: 6
CLUSTER-3 size: 27
Silhouette: [0.329522, 0.341279, 0.678252, stotal: 0.449685]
```

Σημ: Η παράμετρος -**complete** στο Clustering χρησιμοποιείται για να τυπώσουμε και τα items_id που έχει κάθε κεντροειδές. Δεν χρειάζεται κάποια τιμή δίπλα από το όρισμα. Αντίστοιχα, η παράμετρος -**silhouette** χρειάζεται να συμπεριληφθεί για να την προαιρετική εκτύπωση του δείκτη εσωτερικής αξιολόγησης της συσταδοποίησης.

Σε κάθε εκτέλεση μπορούμε να βάλουμε το όνομα ενός output file προκειμένου να τυπωθούν σε αυτό τα αποτελέσματα. Δεν αποτελεί υποχρεωτικό όρισμα και η έξοδος τυπώνεται by default στο αρχείο “output.txt” σε κάθε αλγόριθμο.
Επίσης, η σειρά των παραμέτρων δεν έχει σημασία.

Δομή αρχείων

- **Part_A/Common/ :** Κοινά αρχεία για την λειτουργία όλων των αλγορίθμων, όπως main.cpp, utils.cpp/hpp, H_hash.cpp/hpp, G_hash.cpp/hpp.
- **Part_A/Discrete_Frechet/ :** Περιέχει τα αρχεία για την αναπαράσταση και τον υπολογισμό των καμπυλών με την διακριτή απόσταση Frechet.
- **Part_A/Continuous_Frechet/ :** Περιέχει τα αρχεία για την αναπαράσταση και τον υπολογισμό των καμπυλών με την συνεχόμενη απόσταση Frechet.
- **Part_A/LSH/ :** Περιέχει τα αρχεία για την αναπαράσταση και τον υπολογισμό των διανυσμάτων στον R^d χώρο με lsh και υπολογισμό αποστάσεων μέσω της ευκλείδιας απόστασης.
- **Part_A/Hypercube/ :** Περιέχει τα αρχεία για την αναπαράσταση και τον υπολογισμό των διανυσμάτων στον R^d χώρο με χρήση της προβολής στον υπερκύβο μέσω της ευκλείδιας απόστασης.
- **Part_A/Unit_Testing/ :** Περιέχει το testing αρχείο για τον έλεγχο μικρών κομματιών του υποσυνόλου του προγράμματος, καθώς και έλεγχο της γενικής λειτουργικότητας του.

- **Part_B/Common/ :** Κοινά αρχεία για την λειτουργία όλων των αλγορίθμων, όπως main.cpp, utils.cpp/hpp, H_hash.cpp/hpp, G_hash.cpp/hpp, Point_frechet.cpp/hpp.
- **Part_B/Curve/ :** Περιέχει τα αρχεία για την αναπαράσταση και τον υπολογισμό των καμπυλών με την διακριτή απόσταση Frechet.
- **Part_B/Vector/ :** Περιέχει τα αρχεία για την αναπαράσταση και τον υπολογισμό των διανυσμάτων με χρήση LSH και Hypercube μέσω της ευκλείδιας απόστασης.
- **Part_B/Unit_Testing/ :** Περιέχει το testing αρχείο για τον έλεγχο μικρών κομματιών του υποσυνόλου του προγράμματος, καθώς και έλεγχο της γενικής λειτουργικότητας του.

Περιγραφή υλοποίησης

Στο **Α) μέρος** της εργασίας η ροή του προγράμματος είναι ως εξής:
Αρχικά, διαβάζονται και ελέγχονται τα ορίσματα που δόθηκαν και στη συνέχεια διαβάζεται το input file και αποθηκεύεται κάθε γράμμη του αρχείου σε μία καμπύλη. Ανάλογα με τον αλγόριθμο που επιλέχθηκε να εκτελεστεί θα αρχικοποιηθούν οι ανάλογες δομές όπως τα hashtables και κάποιες δομές vector για την αποθήκευση των δεδομένων. Για την τιμή του delta που χρησιμοποιείται κατά το snapping, πειραματιστήκαμε αρκετά και για το συγκεκριμένο dataset που μας δόθηκε καταλήξαμε ότι το βέλτιστο είναι τιμές που κυμαίνονται στο εύρος [1, 1.5]. Κατόπιν, γίνεται η σταδιακή μετατροπή των καμπυλών σε διανύσματα για την αποθήκευσή τους στις δομές hashtables. Τα βήματα που ακολουθούνται για το Aii) ερώτημα είναι να προσθεθεί ο παραγόντας x του χρόνου, έτσι ώστε οι καμπύλες να ανήκουν στον R^2 και στην συνέχεια γίνεται προβολή των καμπυλών σε δισδιάστατο πλέγμα (snapping) σύμφωνα με τον τύπο που ανακοινώθηκε στο μάθημα. Το LSH σε αυτή την περίπτωση είναι η προβολή στο πλέγμα. Έπειτα από τις καμπύλες θα αφαιρεθούν τα συνέχομενα διπλότυπα και θα γίνει concatenate και padding για να προκύψουν τα vectors διπλάσιου μεγέθους από το αρχικό για να γίνει το hash τους στα L hashtables. Όσο αναφορά το Aiii) ερώτημα όπου εκεί είμαστε στον χώρο μία διάστασης R ακολουθούνται τα εξής βήματα: Αρχικά, γίνεται το φιλτραρισμα των σημειων και απαλείφονται τα σημεία που απέχουν περισσότερο από μία τιμή έψιλον που την ορίζουμε στο πρόγραμμα. Έπειτα, γίνεται η αντίστοιχη προβολή σε μονοδιάστατο πλέγμα και εφαρμόζεται η τεχνική minima-maxima, στην οποία παίρνουμε τις συντεταγμένες ανά τριάδες και απαλείφεται το μεσαίο στοιχείο κάθε φορά αν είναι ανάμεσα στις άλλες δύο τιμές. Στο τελευταίο βημα θα γίνει concatenate και padding για να προκύψουν τα vectors διπλάσιου μεγέθους από το αρχικό για να γίνει το hash τους στο 1 hashtable που χρησιμοποιείται. Στο τέλος, είναι η σειρά των queries να διαβαστούν και να απαντηθούν.Καλείται η συνάρτηση print_query_results() για να τυπώσει τα τελικά αποτελέσματα και έμμεσα καλούνται οι μέθοδοι για αναζήτηση approximate nearest neighbor και bruteforce nearest neighbor, έτσι ώστε να τυπωθούν τα στατιστικά που χρειάζονται. Για την αναζήτηση των κοντινόνερων γειτόνων χρησιμοποιήθηκε η τεχνική με τον έλεγχο των ID.

Στο **Β) μέρος** της εργασίας ανάλογα με τον τρόπο που επιλέγεται από τις παραμέτρους κατά την εκτέλεση του προγράμματος θα γίνει το αντίστοιχο update των κεντροειδών και η ανάθεση των καμπυλών. Αρχικά, γίνεται η αρχικοποίηση των K κεντροειδών, τα οποία αποτελούνται από σημεία μέσα από το input dataset που μας δίνεται. Για την αρχικοποίηση έχουμε ακολουθήσει την μέθοδο Initialization++ σύμφωνα με τις διαφάνειες. Το πρώτο κέντρο επιλέγεται τυχαία με ομοιόμορφη κατανομή και κατόπιν τα υπόλοιπα με βάση πιθανοτήτων σύμφωνα με το τετράγωνο της απόστασης. Για να αποφύγουμε να έχουμε πολύ μεγάλους αριθμούς κάνουμε κανονικοποίηση με έναν σταθερό ακέραιο αριθμό.
Για τις μεθόδους LSH και Hypercube για την παραγωγή των h_i χρησιμοποιήσαμε την μεταβλητή w με μία σταθερή τιμή (w=100), καθώς παρατηρήσαμε ότι έδινε καλύτερα αποτελέσματα. Έπειτα, εκτελείται η διαδικασία της ανάθεσης σημείων και σε κάθε

επανάληψη ανανεώνονται τα κέντρα έως ότου πραγματοποιηθεί το κριτήριο σύγκλισης ή φτάσει σε ένα άνω όριο επαναλήψεων. Για όλες τις αλγοριθμικές μεθόδους το κριτήριο σύγκλισης που έχουμε εφαρμόσει είναι ότι κάθε διάσταση των νέων κέντρων με τα ακριβώς προηγούμενα του δεν πρέπει να απέχουν περισσότερο από έναν αριθμό έψιλον. Για την τιμή του έψιλον που χρειάζεται να χρησιμοποιηθεί για το συγκεκριμένο dataset πειραματιστήκαμε με δοκιμές μέχρι να καταλήξουμε σε κάποια βέλτιστη τιμή. Για την μέθοδο mean curve της διαδικασίας του update των κεντροειδών, βρίσκουμε αρχικά την Frechet απόσταση που επιστρέφεται από τον δισδιάστατο πίνακα και κάνουμε backtracking για να βρούμε ένα optimal traversal. Στην συνέχεια, με βάση το optimal traversal θα αναθέσουμε τις αντίστοιχες καμπύλες στα φύλλα ενός complete binary tree και θα υπολογισούμε ανά δύο τις καμπύλες των φύλλων. Κάθε νέα καμπύλη που προκύπτει από τα φύλλα θα τοποθετηθεί στον κόμβο-γονέα μέχρι να φτάσουμε στην ρίζα και να πάρουμε το τελικό mean curve που θα γίνει το νεό κεντροειδές.Σε κάθε βήμα αύτης της διαδικασίας εκτελούμε filtering έτσι ώστε το μέγεθος της mean_curve να μην μεγαλώνει δραματικα.
Σε κάθε ενημέρωση των νέων κέντρων, αποθηκεύουμε σε ένα vector από int, τα κλειδια των buckets των hashtables που έχουμε ήδη επισκεφθεί για να αρχικοποιήσουμε μόνα τα points σε εκείνα τα buckets.

Παρατηρήσεις – Σχόλια

- Χρησιμοποιήθηκε το προγραμματιστικό εργαλείο git για την ομαλή συνεργασία των μελών της ομάδας.
- Σε κάθε αρχείο υπάρχουν τα απαραίτητα επεξηγηματικά σχόλια στα σημεία που έχει κριθεί απαραίτητο, τα οποία διευκολύνουν την ανάγνωση και κατανόηση του κώδικα.
- Έγινε μεταγλώττιση και εκτέλεση του προγράμματος σε διάφορα μηχανήματα linux της σχολής (όπως linux03, linux16, linux19, linux28).di.uoa.gr, καθώς και επιπλέον δοκιμές λαμβάνοντας υπ’όψιν να λειτουργεί σωστά το πρόγραμμα για κάθε δυνατή περίπτωση στο πλαίσιο της εργασίας.
- Με την βοήθεια του προγραμματιστικού εργαλείου Valgrind έχει ελεγχθεί ότι δεν υπάρχουν errors και memory leaks, όλος ο χώρος αποδεσμεύεται κανονικά.