α) Ανάπτυξη Λογισμικού για Αλγοριθμικά Προβλήματα - Εργασία 2

Η εργασία που αναπτύχθηκε αποτελείται από δύο μέρη. Το πρώτο περιέχει τους αλγόριθμους για την εύρεση κοντινότερου γείτονα. Έχουν υλοποιηθεί και τα τρία ερωτήματα της εκφώνησης, συγκεκριμένα δηλαδή: α) αναπαράσταση των χρονοσειρών ως διανύσματα και εύρεση κοντινότερου γείτονα με LSH και Hypercube (με μετρική L2), β) αναπαράσταση των χρονοσειρών ως πολυγωνικές καμπύλες στο R2 και εύρεση κοντινότερου γείτονα με LSH με βάση την μετρική Discrete Frechet και γ) αναπαράσταση των χρονοσειρών ως πολυγωνικές καμπύλες στην ευθεία R και εύρεση κοντινότερου γείτονα με LSH με βάση την μετρική Continuous Frechet. Το πρόγραμμα για τον κάθε αλγόριθμο, αφού διαβάσει το αρχείο εισόδου (input file) θα δημιουργήσει τις απαραίτητες δομές που χρειάζονται για την αναζήτηση. Στους αλγόριθμους για Frechet γίνονται οι απαραίτητες μετατροπές για την αποθήκευση των καμπυλών στο LSH table. Στην συνέχεια για κάθε query point του αρχείου αναζήτησης (query file) θα βρει τον προσεγγιστικά κοντινότερο γείτονα μέσω του επιλεγμένου αλγορίθμου καθώς και τον πραγματικά κοντινότερο γείτονα μέσω εξαντλητικής αναζήτησης και θα εκτυπώσει τα ζητούμενα (αποστάσεις, χρόνους κ.τ.λ.) στο αρχείο εξόδου (output file).

Το δεύτερο μέρος αφορά τον αλγόριθμο KMeans για την συσταδοποίηση αντικειμένων. Έχουν υλοποιηθεί και οι πέντε συνδυασμοί της εκφώνησης. Η αρχικοποίηση των κεντροειδών πραγματοποιείται με την τεχνική Initialization++. Στην συνέχεια ανάλογα με τον αλγόριθμο ανάθεσης που έχει επιλεχθεί πραγματοποιείται το σχετικό βήμα. Παρέχονται οι επιλογές: Classic (Ανάθεση με Lloyd’s, για διανύσματα και καμπύλες), LSH_Frechet (Αντίστροφη ανάθεση με LSH για καμπύλες), LSH (Αντίστροφη ανάθεση με LSH για διανύσματα) και Hypercube (Αντίστροφη ανάθεση με υπερκύβο για διανύσματα). Μετά την ανάθεση, ανάλογα με την επιλογή του χρήστη εκτελείται το βήμα ανανέωσης. Παρέχονται οι επιλογές Mean Frechet (για ανανέωση καμπυλών) και Mean Vector (για ανανέωση διανυσμάτων). Αφού ολοκληρωθεί ο αλγόριθμος, εκτυπώνεται το μέγεθος και το κεντροειδές κάθε cluster και στην συνέχεια πραγματοποιείται η μετρική silhouette για την αξιολόγηση του αλγορίθμου (σε περίπτωση που ο χρήστης έχει επιλέξει την παράμετρο silhouette). Επίσης εκτυπώνονται τα id των αντικειμένων για κάθε cluster (σε περίπτωση που ο χρήστης έχει επιλέξει την παράμετρο complete).

Σχεδιαστικές επιλογές:

Παραθέτουμε τις παραδοχές και επεκτάσεις υλοποίησης που πήραμε για κάποια τμήματα των προγραμμάτων και αιτιολόγηση αυτών.

1) Παρόλου που η εκφώνηση ζητάει τον πρώτο γείτονα στο Α κομμάτι της εργασίας θα παρατηρήσετε στο πρόγραμμα μας ότι έχουμε αφήσει τον κώδικα της πρώτης εργασίας για την εύρεση Ν γειτόνων (το Ν είναι hardcoded ίσο με 1). Αυτό ήταν μια εσκεμμένη απόφαση μιας και θεωρήσαμε ότι θα έπαιρνε αρκετό χρόνο να αλλάξουμε τα σχετικά κομμάτια. Επίσης αυτό αποτελεί και μια σχεδίαση σε περίπτωση πιθανής επέκτασης υλοποίησης για Ν γείτονες. Το MAF ωστόσο υπολογίζεται λαμβάνοντας υπόψη μόνο τον πρώτο γείτονα.

2) Σχετικά με το Aiii ερώτημα, επειδή παρατηρήσαμε ότι γενικά αργούν πάρα πολύ οι υπολογισμοί πάνω στις μη-φιλτραρισμένες καμπύλες (ιδίως αν οι διαστάσεις είναι πολλές), επιλέξαμε να προσθέσουμε μία ακόμη παράμετρο: -bf_filter. Αυτή είναι μια optional boolean παράμετρος όπου αν την εισάγει ο χρήστης θα γίνουν οι brute force υπολογισμοί με φιλτραρισμένες καμπύλες, με αποτέλεσμα να εκτελεστούν πιο γρήγορα, αλλά με αυξημένη πιθανότητα εύρεσης ενός πιθανού λάθους γείτονα μιας το filtering σε κάποιες καμπύλες αφαιρεί αρκετές διαστάσεις. Οι υπολογισμοί στο querying γίνονται πάντα στις filtered καμπύλες, μιας και εκεί ούτως ή άλλως αποτελεί προσέγγιση η όλη διαδικασία.

3) Στο Aiii ερώτημα το ε που χρησιμοποιείται στο filtering έχει hardcoded τιμή 2.5. Παρατηρήσαμε ότι αυτή η τιμή ήταν ιδανική μιας και δεν καθυστερούσε την εκτέλεση του αλγορίθμου αλλά ούτε επηρέαζε ιδιαιτέρως τα αποτελέσματα για τους γείτονες που βρίσκουμε.

4) Το MAF που ζητείται στο Α κομμάτι της εργασίας ενδέχεται να έχει απροσδιόριστη τιμή. Αυτό μπορεί να συμβεί κυρίως άμα υπάρχουν διανύσματα/καμπύλες ίδια με το query file στο input. Έτσι μπορεί να τύχει να είναι όλες οι true αποστάσεις 0. Σε περίπτωση που γίνει αυτό εκτυπώνεται σχετικό μήνυμα.

5) Το TableSize του LSH έχει τιμή N/8, και στην αναζήτηση γειτόνων και στο clustering. Δοκιμάσαμε γενικά και άλλες τιμές όπως N/4 ή N/16. Το N/8 έδινε γενικά τα πιο γρήγορα και πιο ικανοποιητικά αποτελέσματα.

6) Το w έχει την τιμή 400 όπου έχει χρησιμοποιηθεί. Παρατηρήσαμε με διάφορες εκτελέσεις και ξεχωριστά input αρχεία ότι αυτή η τιμή είναι ιδανική για το w, μιας και δίνει αρκετά ικανοποιητικά αποτελέσματα αλλά και δεν καθυστερεί ιδιαιτέρως την εκτέλεση των αλγορίθμων.

7) Στο δ προσθέσαμε default τιμή 1 και στο Α και Β κομμάτι της εργασίας. Αυτή η τιμή έδινε γενικά τα πιο ικανοποιητικά αποτελέσματα για τα αρχεία που δοκιμάσαμε. Επίσης προσθέσαμε και το δ ως παράμετρο στο Β/clustering παρόλου που δεν το ζητούσε η εκφώνηση.

8) Το M, δηλαδή η τιμή που χρησιμοποιείται για padding στους Frechet αλγορίθμους επιλέγεται τυχαία με ομοιόμορφη κατανομή στο διάστημα [100000, INT_MAX-1000000]. Γενικά μιας και τα πιο πολλά input που είδαμε και δοκιμάσαμε ήταν για τιμές χρηματιστηρίου θεωρήσαμε ότι είναι μια ασφαλή υπόθεση ότι δεν υπάρχουν εξαψήφιες τιμές μετοχών. Από το INT_MAX αφαιρούμε βέβαια ένα σχετικά σημαντικό ποσοστό (1000000) για αποφυγή πιθανών overflows.

9) Τα βήματα της ανάθεσης και ανανέωσης στο clustering εκτελούνται συνολικά για 7 επαναλήψεις. Δοκιμάσαμε και μικρότερους ή μεγαλύτερους αριθμούς επαναλήψεων αλλά ο επιλεγμένος αριθμός έδινε ικανοποιητικά αποτελέσματα σε ιδανικούς χρόνους.

10) Το βήμα της αντίστροφης ανάθεσης και στο LSH και στον υπερκύβο σταματάει άμα ανατεθούν όλα τα στοιχεία του input. Ωστόσο, μπορεί και να διακοπεί όταν σε μία επανάληψη δεν έχει γίνει καμία ανάθεση σε πάνω από τους μισούς clusters. Θεωρήσαμε ότι αυτό είναι

ένα ιδανικό κριτήριο διακοπής για το βήμα της ανάθεσης, μιας και βελτίωνε την χρονική του απόδοση. Τα παραπάνω ισχύουν και για το LSH_Frechet.

11) Στο Β ερώτημα για το Mean Frechet βήμα ανανέωσης εκτελείται filtering πάνω στις μέσες καμπύλες που καταλήγουν σε κάθε κόμβο του δέντρου. Το ε ξεκινάει με τιμή 0.001 και αυξάνεται με αυτό το βήμα μέχρι η μέση καμπύλη να έρθει στην αρχική διάσταση του input αρχείου. Αυτή η επιλογή είδαμε ότι ήταν αρκετά αποτελεσματική μιας και επιτάχυνε ιδιαίτερα την εκτέλεση του βήματος (μέχρι πριν κάναμε filtering μόνο στην ρίζα). Επίσης η τιμή του ε που επιλέξαμε ήταν ιδανική μιας και δεν επηρέαζε πολύ την τελική μέση καμπύλη. Σε αυτό το συμπέρασμα καταλήξαμε με πειραματισμό μιας και είδαμε ότι για μεγαλύτερες τιμές του ε κατέληγε μια μέση καμπύλη με πολλά στοιχεία κομμένα στην αρχή της, και άρα το κεντροείδες δεν ήταν αρκετά ισορροπημένο με αποτέλεσμα να καταλήγουμε με κενούς clusters. Με αυτή την τιμή το ε δίνει πίσω ισορροπημένες μέσες καμπύλες και άρα καλύτερα silhouette scores.

β) Κατάλογος αρχείων κώδικα/επικεφαλίδων και περιγραφή τους

SearchMain.cpp // Αποτελεί την main συνάρτηση που χρησιμοποιείται για την εκτέλεση των αλγορίθμων του Α κομματιού της εργασίας.

VectorElement.cpp // Αρχείο κώδικα της κλάσης VectorElement. Η κλάση αυτή χρησιμοποιείται για την αναπαράσταση ενός διανύσματος. Αποτελείται από έναν πίνακα με τις συντεταγμένες του, το id του και άλλες χρήσιμες πληροφορίες για αυτό.
VectorElement.h Αρχείο επικεφαλίδας της κλάσης VectorElement.

ClusterElement.cpp // Αρχείο κώδικα της κλάσης ClusterElement. Η κλάση αυτή χρησιμοποιείται για την αναπαράσταση μιας καμπύλης. Παρέχει πεδία για την αναπαράσταση της καμπύλης και στους R2 και R χώρους καθώς και όλες τις τροποποιήσεις που μπορεί να έχει (filtered, grid, κ.τ.λ.).
ClusterElement.h // Αρχείο επικεφαλίδας της κλάσης ClusterElement.

DiscreteFrechet.cpp // Αρχείο κώδικα για τις συναρτήσεις σχετικά με το Discrete Frechet (υπολογισμός απόστασης, optimal traversal κ.τ.λ.).
DiscreteFrechet.h // Δηλώσεις για τις σχετικές συναρτήσεις.

LSHash.cpp // Αρχείο κώδικα της κλάσης LSHash. Η κλάση αυτή αποτελεί ένα Local Sensitive Hash table και όλες τις απαραίτητες συναρτήσεις για την εισαγωγή αντικειμένων και αναζήτηση αυτών.
LSHash.h // Αρχείο επικεφαλίδας της κλάσης LSHash.

HyperCube.cpp // Αρχείο κώδικα της κλάσης HyperCube. Η κλάση αυτή αποτελεί έναν υπερκύβο (υλοποιημένο ως hash table) και όλες τις απαραίτητες συναρτήσεις για την εισαγωγή αντικειμένων και αναζήτηση αυτών.
HyperCube.h // Αρχείο επικεφαλίδας της κλάσης HyperCube.

TableF.cpp // Αρχείο κώδικα της κλάσης TableF. Η κλάση αυτή χρησιμοποιείται για την αποθήκευση τιμών της συνάρτησης f στον υπερκύβο. Αποτελείται από ένα hash table όπου αποθηκεύει όσες τιμές, μιας h, έχουν πάρει 0 και όσες έχουν πάρει 1. Κάθε συνάρτηση f έχει από ένα δικό της TableF.
TableF.h // Αρχείο επικεφαλίδας της κλάσης TableF.

Neighbours.cpp // Αρχείο κώδικα της κλάσης neighboursInfo. Η κλάση αυτή χρησιμοποιείται για την συγκέντρωση N γειτόνων για ένα query. Αποτελείται από έναν πίνακα με τα υποψήφια id καθώς και έναν πίνακα με τις αποστάσεις που συγκεντρώθηκαν. Κάθε LSHash ή ενα HyperCube έχει έναν πίνακα μεγέθους query rows, όπου κάθε θέση του αντιστοιχεί σε ένα αντικείμενο neighboursInfo, για κάθε query.
Neighbours.h // Αρχείο επικεφαλίδας της κλάσης neighboursInfo.

IdDistancePair.cpp // Αρχείο κώδικα της κλάσης idDistancePair. Η κλάση αυτή περιέχει το id και την απόσταση ενός υποψήφιου γείτονα. Χρησιμοποιείται στις main συναρτήσεις για την κατάλληλη οργάνωση των δεδομένων που θέλουμε στο output.
IdDistancePair.h // Αρχείο επικεφαλίδας της κλάσης idDistancePair.

ClusterMain.cpp // Αποτελεί την main συνάρτηση που χρησιμοποιείται για την εκτέλεση των συνδυασμών του Β κομματιού της εργασίας.

KMeans.cpp // Αρχείο κώδικα της κλάσης KMeans. Η κλάση αυτή περιέχει όλα τα απαραίτητα δεδομένα για την εκτέλεση του αλγορίθμου KMeans όπως αριθμό και πίνακα από clusters, HyperCube αντικείμενο και πίνακα από LSHash αντικείμενα για την αντίστροφη ανάθεση κ.τ.λ. Οι μέθοδοι της κλάσης αυτής αντιστοιχούν στα βήματα του αλγορίθμου, όπως αρχικοποίηση κεντροϊδων, ανάθεση αντικειμένων, ανανέωση κ.τ.λ.
KMeans.h // Αρχείο επικεφαλίδας της κλάσης KMeans.

Cluster.cpp // Αρχείο κώδικα της κλάσης Cluster. Η κλάση αυτή περιέχει δεδομένα από ένα cluster όπως ένα VectorElement/CurveElement (ανάλογα το βήμα ανανέωσης) αντικείμενο που αναπαριστά το κεντροειδές του, λίστα από VectorElement/CurveElement για τα στοιχεία που ανήκουν στον cluster κ.τ.λ.
Cluster.h // Αρχείο επικεφαλίδας της κλάσης Cluster.

TreeNode.cpp // Αρχείο κώδικα της κλάσης TreeNode. Η κλάση αυτή χρησιμοποιείται για την αναπαράσταση του complete binary tree που απαιτεί το βήμα ανανέωσης στο Mean Frechet.
TreeNode.h // Αρχείο επικεφαλίδας της κλάσης TreeNode.

Helpers.cpp // Επιπλέον βοηθητικές συναρτήσεις, όπως και για χρήση debugging κ.τ.λ.
Helpers.h // Δηλώσεις των σχετικών συναρτήσεων.

UnitTestMain.cpp // Αποτελεί την main συνάρτηση για τα unit tests που έχουν ζητηθεί. Περισσότερα για τα unit tests στην τελευταία παράγραφο, στ).

Ο φάκελος fred περιέχει τα αρχεία για την ενσωμάτωση της Continuous Frechet απόστασης. Τα αρχεία αυτά έχουν υποστεί ελάχιστες τροποποιήσεις για να τα ενσωματώσουμε στο

πρόγραμμα μας. Συγκεκριμένα σχολιάσαμε ό,τι είχε σχέση με την Python και προσθέσαμε το #include <algorithm> στο frechet.cpp. Η συνάρτηση μας που αξιοποιεί την Continuous Frechet είναι στο αρχείο Helpers.cpp και είναι η ret_CFD.

γ) Το πρόγραμμα μεταγλωττίζεται με την χρήση της εντολής: make

Επίσης παρέχεται η δυνατότητα του να αφαιρέσει κανείς τα object και executable αρχεία που παράγονται από το Makefile με την χρήση της εντολής: make clean

Να σημειώσουμε επίσης ότι χρησιμοποιήσαμε γενικά το -O2 flag με αποτέλεσμα αρκετά πράγματα να εκτελούνται πιο γρήγορα.

δ) Οδηγίες χρήσης του προγράμματος

Α) Οι αλγόριθμοι για την εύρεση κοντινότερων γειτόνων μπορούν να κληθούν με το εκτελέσιμο: ./search

Παρέχονται οι επιλογές του να εκτελέσει κανείς τον αλγόριθμο χωρίς ορίσματα (όπου εκεί θα χρησιμοποιηθούν οι default παράμετροι αλλά θα ζητηθεί από τον χρήστη να θέσει τα ονόματα των αρχείων) ή με την χρήση των παραμέτρων που αναφέρει η εκφώνηση. Ακολουθούν κάποιες ενδεικτικές εκτελέσεις, μια για κάθε αλγόριθμο:

./search -i nasd_input.csv -q nasd_query.csv -k 5 -L 5 -o myLogFile.txt -algorithm LSH
./search -i nasd_input.csv -q nasd_query.csv -k 5 -M 70 -probes 25 -o myLogFile.txt -algorithm Hypercube
./search -i nasd_input.csv -q nasd_query.csv -k 5 -L 5 -o myLogFile.txt -algorithm Frechet -metric discrete -delta 1
./search -i nasd_input.csv -q nasd_query.csv -k 5 -o myLogFile.txt -algorithm Frechet -metric continuous -delta 1 -bf_filter

Για κάθε μία από τις παραπάνω εκτελέσεις παρέχονται ενδεικτικά output files στο παραδοτέο (lsh_l2.txt, cube_l2.txt, lsh_dfd.txt και lsh_cfd.txt)

Β) Οι συνδυασμοί για το clustering μπορούν να κληθούν με το εκτελέσιμο: ./cluster

Παρέχονται οι επιλογές του να εκτελέσει κανείς τον αλγόριθμο χωρίς ορίσματα (όπου εκεί θα χρησιμοποιηθούν οι default παράμετροι αλλά θα ζητηθεί από τον χρήστη να θέσει τα ονόματα των αρχείων) ή με την χρήση των παραμέτρων που αναφέρει η εκφώνηση. Ακολουθούν κάποιες ενδεικτικές εκτελέσεις, μια για κάθε συνδυασμό:

./cluster -i nasd_input.csv -c cluster.conf -o myLogFile.txt -update Mean Vector -assignment Classic -silhouette -complete
./cluster -i nasd_input.csv -c cluster.conf -o myLogFile.txt -update Mean Vector -assignment LSH -silhouette -complete
./cluster -i nasd_input.csv -c cluster.conf -o myLogFile.txt -update Mean Vector -assignment Hypercube -silhouette -complete
./cluster -i nasd_input.csv -c cluster.conf -o myLogFile.txt -update Mean Frechet -assignment Classic -silhouette -complete

./cluster -i nasd_input.csv -c cluster.conf -o myLogFile.txt -update Mean Frechet -assignment LSH_Frechet -silhouette -complete

Για κάθε μία από τις παραπάνω εκτελέσεις παρέχονται ενδεικτικά output files στο παραδοτέο (cluster_classic_vector.txt, cluster_lsh_vector.txt, cluster_cube_vector.txt, cluster_classic_frechet.txt και cluster_lsh_frechet.txt)
Επίσης έχουμε παραδώσει και το cluster.conf που χρησιμοποιήθηκε για τις εκτελέσεις. Ως inputs χρησιμοποιήσαμε τα αρχεία εξέτασης του e-class.

ε) Ονοματεπώνυμα και ΑΜ:
Παντελεήμων Μαλέκας 1115201600268
Θεοφάνης Μπιρμπίλης 1115201600110

στ) Unit Tests

Το Makefile παράγει ένα ακόμη εκτελέσιμο για την εκτέλεση των unit tests που έχουν ζητηθεί. Αυτό είναι το utest και εκτελείται χωρίς ορίσματα (./utest).

Χρησιμοποιήσαμε το framework CppUnit για τα unit tests. Για την εγκατάσταση του βασιστήκαμε στο link:
https://blog.birost.com/a?ID=00150-fef8810b-6a92-4613-ad66-83146164be82

Ενδέχεται το βήμα 3 για το library path να μην είναι απαραίτητο αλλά εμείς το χρειαστήκαμε. Για να συνδέσουμε λοιπόν την shared library της CppUnit ακολουθήσαμε την επιλογή 2) του συνδέσμου.

Οι συναρτήσεις που χρησιμοποιήσαμε για testing ήταν η checkItem της TableF κλάσης και οι hammingDistance και binarySearch του Helpers.cpp. Ξέρουμε ότι γενικά το unit testing προτιμάται να γίνεται πάνω σε μεθόδους από κλάσεις, αλλά οι περισσότερες συναρτήσεις που είχαμε ήταν void τύπου ή χρησιμοποιούσαν κατανομές με αποτέλεσμα να μην έχουμε βέβαιο αποτέλεσμα κ.τ.λ. Με αυτές τις συναρτήσεις ήταν εύκολο να κρίνουμε το αποτέλεσμα.

Αφού γίνουν λοιπόν τα unit tests εκτυπώνονται τα αποτελέσματα στην γραμμή εντολών καθώς και σε ένα xml αρχείο. Το xml αρχείο έχει όνομα Project2UnitTests.xml και το έχουμε παραδώσει και αυτό.

Η εργασία αναπτύχθηκε με χρήση Git και είναι διαθέσιμη στο repository: https://github.com/pantmal/Algorithmic-Problems-Project-2. Επίσης χρησιμοποίηθηκε και το extension Live Share του VS Code για την ταυτόχρονη συγγραφή κώδικα.