2η Εργασία 2021 - 2022

Συνεργάτες:

- **Βησσαρίων Μουτάφης , A.M:1115201800119, mail: sdi1800119@di.uoa.gr**
- **Αρίστη Παπασταύρου , A.M:1115201800154, mail: sdi1800154@di.uoa.gr**

1. Example Section

NOTE: Είναι απαραίτητο πριν από κάθε make, να κάνετε πρώτα make clean και μετά οποιοδήποτε make, μιας και από αλγόριθμο σε αλγόριθμο έχουμε κάποια διαφορετικά directives τα οποία μπορεί να μην κάνουν compile μέρος του κώδικα.

Για το compilation και run της εφαρμογής παρέχουμε ένα Makefile στο root directory. Οι πιθανές εντολές είναι:

```
~$ make search [verbose=1] [debug=1] # make the search executable
~$ In order to print the accuracy of each of the above algorithms in the
    output file you have to type the above commands like this: make search
    verbose=1
~$ make clustering [verbose=1] [debug=1] # make the clustering test case
~$ make tests [verbose=1] [debug=1] # create test cases
~$ ./run_tests # run the unit tests
~$ make clean # delete objective files
```

2. Κατάλογος αρχείων πηγαίου κώδικα

Στο directory src θα βρείτε τα αρχεία:

- Curves.cpp
- GenericClusterSolver.cpp
- NearestNeighboursSolver.cpp
- Point.cpp
- rules.inc
- src.inc

Και στα subfolders που υπάρχουν μέσα στο src:

Στο directory Clustering:

- AssignmentStepRoutines.cpp
- clustering_main.cpp
- KMeans_pp_Solver.cpp

Στο directory CurveCluster:

- CurveAssignmentStep.cpp
- CurveCluster.cpp
- curveclustering.mk

Στο directory CurveLSH:

- curve_lsh.cpp
- CurveNearestNeighboursSolver.cpp
- GenericCurveNearestNeighboursSolver.cpp
- lsh.mk

Στο directory HyperCube:

- hypercube.cpp

Στο directory LSH:

- LSHNearestNeighbours.cpp

Στο directory utils:

- ArgumentParser.cpp
- CurveHashing.cpp
- Evaluator.cpp
- FileHandler.cpp
- Hashing.cpp
- Profiler.cpp
- Utilities.cpp

Στο directory /utils/Fred:

- config.cpp
- frechet.cpp
- interval.cpp
- testmain.cpp

3. Time Series as Points - kNN and Clustering

Στην περίπτωση όπου η αναπαράσταση των time series είναι με $k-D$ points, τότε χρησιμοποιούμε τους αλγορίθμους της 1ης εργασίας γα τους οποίους συμπεριλαμβάνουμε το documentation και ενα experimental notes sum up στο *README1.pdf.*

4. Curve Hashing

Στο curve hashing στόχος είναι να λάβουμε ως είσοδο μια καμπύλη και να επιστρέψουμε το grid curve της. Ο αλγόριθμος ξεκινά κάνοντας iteration ξεχωριστά σε κάθε point του curve. Για κάθε point βρίσκουμε το hash του με βάση τον τύπο των διαφανειών:

$$\boldsymbol{floor((point.getCoordinate(i) - t[i])/delta + 0.5) * delta + t[i]}$$

Ταυτόχρονα κρατάμε και μια δομή previousMinPoint, το οποίο μας βοηθά να ελέγχουμε αν έχουμε duplicates.Το σύνολο των hashes από όλα τα points του curve, τα κάνουμε concatenate στο gridCurve, το οποίο και κάνουμε return.

4.1 Hashing Curve Base Interface

Η Curve Hashing Class, είναι ένα interface για τις hashing classes του Discrete/Continuous LSH for curves.Υλοποιεί τις απαραίτητες μεθόδους για τους 2 τύπους της LSH for curves, την *curveHashing*, η οποία αναλαμβάνει το curve snapping στο κατάλληλο translated grid $G_\delta^t = \{(a_1\delta+t)\ ...\ (a_d\delta+t) | t \in (0,\delta)^d, \delta \in R^+, \forall a_i \in Z\}$ και κρατάει τις βασικές παραμέτρους που θα χρειαστούν κατα το hashing. Επίσης έχει ενα pure virtual overload του call routine, το οποίο πρέπει να υλοποιήσει καθένα από τα subclasses ωστε να μπορεί κανείς να πραγματοποιήσει το hashing για curves.

4.2 LSH with Discrete Frechet

Για την υλοποίηση του LSH με Discrete Frechet, χρησιμοποιήσαμε ενα subclass της Base Class *HashingCurve*, το *DLHHashingCurve.* Ο αλγόριθμος που υλοποιήσαμε:

- Περνάμε την κάθε καμπύλη από την συνάρτηση *CurveHashing*
- Κάνουμε **squeeze** το grid hash του προηγούμενου βήματος. Δηλαδή κάνουμε concatenate τα coordinates του κάθε grid curve point σε ένα point.
- Κάνουμε padding στο τέλος του point τόσα μηδενικά όση είναι η διαφορά:

$$dim * max(curve\_length) - point.current\_dimension$$

- Επιστρέφουμε το edited point

4.3 LSH with Continuous Frechet

Για την υλοποίηση του LSH με Continuous Frechet, χρησιμοποιήσαμε ενα subclass της Base Class *HashingCurve*, το *CLHHashingCurve.*

Ακόλουθήσαμε την θεωρία και υλοποιήσαμε τον αλγόριθμο για το continuous curve hashing ως εξής: Για κάθε ένα curve $c \in Dataset$

- κάνουμε filtering με την *CLSHHashing::filter*, ώστε να προβάλουμε την καμπύλη στο R,
- χρησιμοποιούμε την ίδια συνάρτηση για grid snapping όπως και στο discrete curve hashing, ώστε να κάνουμε map τα σημεία της filtered καμπύλης c στα σημεία του transleted-grid $G_\delta^t = \{(a_1\delta + t) \ \dots \ (a_d\delta + t) | t \in (0,\delta)^d, \delta \in R^+, \forall a_i \in Z\}$, σύμφωνα με τον τύπο
$$c' = \lfloor \frac{(x-t)}{\delta} + \frac{1}{2} \rfloor \cdot \delta + t$$
- στο grid curve κρατάμε μόνο τα τοπικά μέγιστα και ελάχιστα
- φτιάχνουμε το concatenated curve vector, x
- εφαρμόζουμε padding βάσει της μεγαλύτερης τιμής απ'όλα τα grids

Η κλάση αυτή περιέχει και την μέθοδο filtering, η οποία με βάσει ένα error κάνει filter το δεδομένο curve. Για περεταίρω χρήση της μεθόδου αυτής διαβάστε και το section *Clustering::Mean Curve Update*

4.4 Generic Solver

O generic solver υλοποιήθηκε αποκλειστικά για να μπορούμε να χρησιμοποιούμε οποιαδήποτε από τις 2 μορφές Curve Hashing (Discrete/Continuous) με τον ίδιο evaluator λόγω των virtual συναρτήσεων που υλοποιήσαμε.

5. kNN on Time Series

Για το clustering των time series σε αναπαράσταση με curves, χρησιμοποιήσαμε τους ίδιους αλγορίθμους για reverse assignment και lloyd assignment, με την μετατροπή των αλγορίθμων της πρώτης εργασίας σε μορφή templates (δες παρακάτω).

5.1 Generic Solver

O generic cluster solver υλοποιήθηκε αποκλειστικά για να μπορούμε να χρησιμοποιούμε οποιαδήποτε από τις 2 μορφές clustering με τον ίδιο evaluator λόγω των virtual συναρτήσεων που υλοποιήσαμε.

5.2 Experimental Notes

Παρατηρούμε ότι, για μεγαλύτερες τιμές του L, έχουμε μικρότερο MAF.Για μεγαλύτερες τιμές του k, τα curves διαχωρίζονται πιο αραιά στα buckets λόγω του υψηλού sensitivity ως προς την απόσταση τους οπότε και το search γίνεται πιο αποδοτικό λόγω του μειωμένου search space.Σύμφωνα με τις πειραματικές μετρήσεις που κάναμε πάνω στα δοθέντα datasets, για υψηλότερες τιμές του k και L θα πετύχουμε την καλύτερη εναλλαγή μεταξύ MAF και speedup.

6. Clustering on Time Series

Στο clustering με Mean Vector ακολουθήσαμε την λογική της πρώτης εργασίας εγκαθιστώντας ενα απλό flow στην main ώστε να καθορίζει ποιό solver θα πρέπει να εισάγει και πως να κάνει evaluate το clustering.

6.1 Assignment Templates from Project 1

Στον φάκελο ReverseAssignment.hpp και LLoyd.hpp υπάρχουν templates για να μπορεί ο χρήστης να καλέσει τις συναρτήσεις από την πρώτη εργασία χωρίς να χρειάζεται να διευκρινήσει τύπους. Η προσθήκη templates διευκολύνει την κλήση των αλγορίθμων με την παροχή του dataset των centroids και ενος solver που υλοποιούν κάποιες standard μεθόδους. Τα templates χρησιμοποιήθηκαν για να αποφευχθέι το redudancy στον κώδικα εφόσον οι λειτουργίες έμεναν ίδιες με την πρώτη εργασία.

6.2 Reverse Assignment - Range Search with LSH for curves

Το *reverse assignment* με *lsh with descrete frechet* υλοποιήθηκε ορίζοντας **στην πρώτη κλήση** ένα LSHSolver για τα curves, αρχικοποιόντας το βάσει του dataset και ορίζοντας

6.3 Mean Curve Update

Το update step στον KMeans, προσπελαύνει όλα τα centroids και τα curves που έχουν ανατεθεί σε αυτά. Τα curves του κάθε centroid, αποθηκεύονται σε ένα vector από curves (Curvetree), το οποίο δίνεται σαν παράμετρο στην συνάρτηση εύρεσης του Mean Curve. Αφού υπολογιστεί το Mean Curve, ανατείθεται ως το νέο centroid.

Η συνάρτηση για την εύρεση του Mean Curve έχει υλοποιηθεί σε 3 ιεραρχικά στάδια. Η αρχική συνάρτηση παίρνει σαν όρισμα ένα vector από Curves, όπου στοχεύει στην εύρεση του ολικού mean curve. Αντί όμως να το υλοποιήσουμε με ένα binary complete tree, μεταφέραμε την ίδια λογική σε υλοποίηση με πίνακα. Δηλαδή ο πίνακας είναι γεμισμένος σαν να κάναμε level order traversal σε b-tree. Μέσα στον βασικό αλγόριθμο θα δείτε ότι στο κεντρικό loop προχωράμε κατά 2*step και κάθε φορά τσεκάρουμε το φύλλο αυτό με το φύλλο στην θέση i + step. Τα curves στις θέσεις αυτές τα περνάμε σαν όρισμα στην επόμενη στην ιεραρχία getMeanCurve και αποθηκεύουμε το mean curve στην θέση i ενώ διαγράφουμε το curve στην θέση i+step.Για να διατηρήσουμε το dimensionality,χρησιμοποιούμε τεχνικές filtering παρόμοιες με αυτές που χρησιμοποιούμε στο Continuous Curve LSH, βασιζόμενοι σε ένα error tolerance το οποίο αυξάνεται μέχρι να επιτευχθεί ένα βιώσιμο mean curve size.
Η επόμενη στην ιεραρχία getMeanCurve απλά καλεί την συνάρτηση εύρεσης του optimal path (την οποία προσθέσαμε στην κλάση Frechet του Fred) και υπολογίζει μέσω του optimal path το mean curve μεταξύ 2 οποιονδήποτε curves. Η τελευταία στην ιεραρχία getMeanCurve υπολογίζει με την χρήση της L2_Norm την μεταξύ απόσταση των σημείων του κάθε tuple του optimal path.

6.4 Experimental Notes

Παρατηρούμε πως ο classic curve clustering αλγόριθμος, χρειάζεται πάντα σχεδόν περισσότερο χρόνο για να ολοκληρώσει απ οτι ο πιθανοτικός clustering αλγοριθμος με curve-lsh reverse assignment. Αυτό οφείλεται στο γεγονός πως η μετρική για την μέτρηση αποστάσεων μεταξύ 2 καμπυλών ορίστηκε να είναι η discrete frechet, η οποία ειναι αρκετά χρονοβόρα διαδικασία. περιορίζοντας το search space το reverse assignment μας, *με σωστό tuning*, υπερνικάει τον αλγόριθμο πετυχαίνοντας την κατάλληλη κατανομή των curves στα buckets. Σημαντικό ειναι να τονιστεί πως τα δ, w, υπολογίζονται από το dataset αλλα θα μπορούσαν να γίνουν tune με το χέρι. Επίσης για καλύτερα αποτελέσματα μπορούμε να ανεβάσουμε το L: *αριθμός translated grids,* G_δ^t και το k: *αριθμός* $h(\cdot)$ *συναρτήσεων στον lsh για points (1η εργασία)* ώστε να μπορέσουμε να πετύχουμε πιο εξειδικευμένη κατανομή των concatenated grid curves και μεγαλύτερο accuracy (μικρότερο MAF) με την αύξηση των curve hash functions από την οικογένεια LSH, του grid μας.