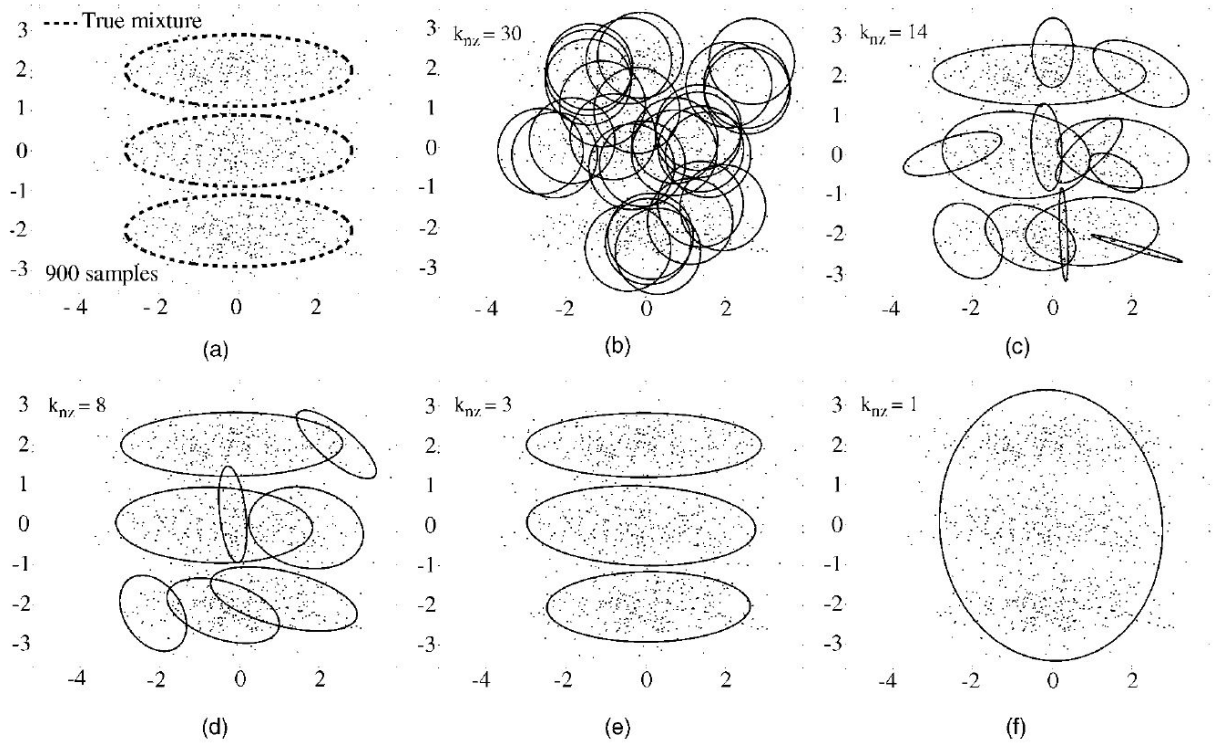
Ανάπτυξη Λογισμικού για Δισεπίλυτα Αλγοριθμικά Προβλήματα

Εργασία 3η - Recommendation System and Applications in Molecular Biology



Αυτσίδης Σέργιος - sdi0900052
Μπορεκτσίογλου Ιωάννης - sdi1000111
Πατσουράκος Κωνσταντίνος - sdi0900129

Επεξήγηση Εργασίας

Γενικές Παρατηρήσεις

Η εργασία υλοποιήθηκε κατά τα ζητούμενα. Τα 2 μέρη υλοποιήθηκαν ως ξεχωριστές εφαρμογές, και παράγονται διαφορετικά εκτελέσιμα.

Για την μεταγλώττιση, παρέχονται makefiles με τα αντίστοιχα make targets που ανταποκρίνονται σε κάθε εφαρμογή. Περαιτέρω πληροφορίες παρέχονται στην συνέχεια, ξεχωριστά για το κάθε μέρος.

Ιεραρχία Αρχείων / Φακέλων

Το παραδοτέο αποτελείται από 2 directories.

1. **recommend:** Περιέχει την εφαρμογή που ζητείται στο πρώτο μέρος, δηλαδή το recommendation system.
2. **molClust:** Περιέχει την εφαρμογή που ζητείται στο δεύτερο μέρος, δηλαδή την εφαρμογή συσταδοποίησης μοριακών διαμορφώσεων. Εσωτερικά περιέχονται 2 directories με διαφορετικές υλοποιήσεις για dRMSD και cRMSD.

Σχετικά με την παραλληλοποίηση

Εκτενής παραλληλοποίηση δεν υλοποιήθηκε. Παρ'ολ'αυτά, ο αλγόριθμος αξιολόγησης clustering Silhouette παραλληλοποιήθηκε με την χρήση OpenMP.

Αναλυτική επεξήγηση

Μέρος 1: Recommendation System

Το μέρος 1 εκτελείται ως μια εφαρμογή. Πρώτα γίνεται recommend των 5 καλύτερων items για κάθε user με τον αλγόριθμο NN-LSH, και στην συνέχεια με Cluster-Recommend. Εαν έχει δοθεί η επιλογή *--validate*, μετά από την εκτέλεση κάθε αλγορίθμου γίνεται validation των αποτελεσμάτων με τον KFoldValidate (k=10).

Η είσοδος και η έξοδος της εφαρμογής ακολουθεί το μοτίβο και τις συμβάσεις που περιγράφονται στις απαιτήσεις της άσκησης.

Μεταγλωττιση - Εκτέλεση:

Για την ευκολότερη μεταγλώττιση του προγράμματος παρέχεται ένα αρχείο `makefile`. Η εντολή μεταγλώττισης συνεπώς είναι

```
% make
```

Για την εκτέλεση του προγράμματος αρκεί μια εντολή της παρακάτω μορφής

```
% ./recommend -d <input> -c <conf> -o <output>
```
,με:

- input: το αρχείο εισόδου, όπως περιγράφεται στην εκφώνηση
- conf: το αρχείο configuration, που δίνει αριθμούς για τον LSH και το ClaransUpdate
- output: το αρχείο εξόδου, όπου το πρόγραμμα παρέχει τις ζητούμενες εκτυπώσεις / πληροφορίες

Ακόμη, με την επιλογή --validate, εκτελείται η μέθοδος 10foldValidation.

A. Recommendation using NN-LSH

1. Γενικά:

Αρχικά η εφαρμογή χτίζει τον LSH με το dataset. Στην συνέχεια, κάνει query search με τον αλγόριθμο που περιγράφεται παρακάτω για όλα τα σημεία του dataset, ορίζοντας την γειτονιά του κάθε σημείου. Στο τέλος επιλέγει τα 5 καλύτερα items για τον κάθε χρήστη και τα εμφανίζει.

2. Αλγοριθμικές επιλογές:

Ο αλγόριθμος εύρεσης των *P* κοντινότερων γειτόνων με την χρήση του LSH ακολουθεί τα παρακάτω βήματα:

i. Αναζήτηση στον LSH με query τον user του οποίου θέλουμε να βρούμε τους γείτονες - ***O(L)***

ii. Deduplication των αποτελεσμάτων και αποθήκευση των αποστάσεων από το query - ***O(resNum*d)***
iii. Ταξινόμηση των users που βρέθηκαν με κριτήριο την απόσταση από το query - ***O(resNum*log(resNum))*** , καθώς οι αποστάσεις είναι **ήδη γνωστές** από το προηγούμενο βήμα
iv. Επιλογή των *P* κοντινότερων αποτελεσμάτων. - ***O(P)***

Συνολική πολυπλοκότητα για κάθε query: ***O(resNum*K)***, K=max{ log(resNum),d }

Ο αλγόριθμος που περιγράφει η εκφώνηση (binary repeated search range) δεν θα λειτουργούσε αποδοτικά στην δική μας υλοποίηση, καθώς ένα range search στον LSH λειτουργεί ως εξής:

i. Εντοπισμός των buckets που αντιστοιχούν στο query σε κάθε hashtable - ***O(L)***
ii. Έλεγχος πόσα στοιχεία του bucket βρίσκονται σε απόσταση από το query μικρότερη από το range - ***O(resNum*d)***

Τα παραπάνω βήματα επαναλαμβάνονται μέχρι να επιτύχουμε ένα range που να μας δίνει P αποτελέσματα. Συνεπώς η πολυπλοκότητα της αναζήτησης είναι της τάξης του ***O(resNum*d*N)***, με N το πλήθος των βημάτων αναζήτησης. Συνεπώς, η πολυπλοκότητα του αλγορίθμου αυτού (με την δικιά μας υλοποίηση) είναι κατά πάσα πιθανότητα χειρότερη από την υλοποίηση που ακολουθήθηκε.

3. Αποτελέσματα εκτελέσεων - Validation

Μια μέση εκτέλεση του αλγορίθμου (επιλέγουμε k=6 και L=5, και input το dataset που δόθηκε) μας δίνει τα παρακάτω αποτελέσματα:

- Χρόνος Recommend: 59.376sec
- Χρόνος Validation: 8m 56.496sec
- Mean Absolute Error: 1.02

B. Cluster-Based Recommendation

1. Γενικά

Η εφαρμογή αρχικά χτίζει το clustering με τιμή k = N/100. Στην συνέχεια, εκτελεί επαναλαμβανόμενες αυξομειώσεις του k, υπολογίζοντας κάθε φορά την τιμή του Silhouette για το συγκεκριμένο k. Σταματάει όταν το k φτάσει κοντά στο N/P. Αυτό το όριο επιλέχθηκε διότι εαν αυξανόταν το k παραπάνω, ο μέσος πληθικός αριθμός των στοιχείων των clusters θα έπεφτε κάτω από N/P, με αποτέλεσμα να μην έχουμε κατά μέσο όρο P γείτονες σε κάθε cluster, και να εκτελείται το recommendation με λιγότερους γείτονες (πιθανώς χειρότερα αποτελέσματα - αδυναμία σύγκρισης της απόδοσης του recommendation ως προς τον αριθμό P των γειτόνων). Στο τέλος, ξαναχτίζει το clustering με την optimal τιμή του k (με βάση τον Silhouette).

Σε αυτό το clustering στην συνέχεια εξετάζουμε κάθε cluster και κάνουμε recommend εντός του cluster με βάση τα items όλων των user του. Τέλος, για κάθε user επιλέγουμε τα 5 items με το μεγαλύτερο score.

2. Αλγοριθμικές επιλογές

Για το clustering επιλέχθηκαν οι παρακάτω αλγόριθμοι:

- initialization: Kmedoids++
- assignment: PamAssign
- update: ClaransUpdate

καθώς μετά από δοκιμές αποφασίσαμε ότι ήταν ο πιο αποδοτικός συνδυασμός τόσο σε ποιότητα clustering όσο και σε χρόνο εκτέλεσης

3. Αποτελέσματα εκτελέσεων - Validation

Μια μέση εκτέλεση του αλγορίθμου (k=189) μας δίνει τα παρακάτω αποτελέσματα:

- Χρόνος: 1m 11.457sec
- Χρόνος Validation: 3m 9.094sec
- Mean Absolute Error: 0.949

Μέρος 2: Molecular Conformation Clustering

Μεταγλωττιση - Εκτέλεση:

Για την ευκολότερη μεταγλώττιση του προγράμματος παρέχεται ένα αρχείο `makefile`. Υπάρχει διαφορετικό αρχείο στους φακελους cRMSD και dRMSD. Η εντολή μεταγλώττισης συνεπώς είναι σε κάθε φάκελο

```
% make
```

Για την εκτέλεση του προγράμματος στο φάκελο cRMSD αρκεί μια εντολή της παρακάτω μορφής

```
% ./molConAPP -i <input> -c <conf> -o <output> -k <clusters>
```

,με:

- input: το αρχείο εισόδου, όπως περιγράφεται στην εκφώνηση
- conf: το αρχείο configuration, που δίνει αριθμούς για τον LSH και το ClaransUpdate
- output: το αρχείο εξόδου, όπου το πρόγραμμα παρέχει τις ζητούμενες εκτυπώσεις / πληροφορίες
- k: πλήθος Clusters

Για την εκτέλεση του προγράμματος στο φάκελο dRMSD αρκεί μια εντολή της παρακάτω μορφής

```
% ./molConAPP -i <input> -c <conf> -o <output> -r <r>
```

,με:

- input: το αρχείο εισόδου, όπως περιγράφεται στην εκφώνηση
- conf: το αρχείο configuration, που δίνει αριθμούς για τον LSH και το ClaransUpdate
- output: το αρχείο εξόδου, όπου το πρόγραμμα παρέχει τις ζητούμενες εκτυπώσεις / πληροφορίες
- r: η τιμή r

Και στις δύο περιπτώσεις παρέχεται η επιλογή -d για default τιμές

1. Γενικά:

Η εφαρμογή υλοποιήθηκε σε 2 ξεχωριστά εκτελέσιμα, με τις 2 distances (cRMSD , dRMSD). Παράγονται 2 διαφορετικά εκτελέσιμα, με τις εντολές (..)

Για τους αλγεβρικούς υπολογισμούς που χρειάστηκαν χρησιμοποιήθηκαν οι LAPACK(E) και BLAS. Υλοποιήθηκε ένα υποτυπώδες interface για να είναι πιο εμφανείς και κατανοητές οι κλησεις τους, στο αρχείο *lapackeIF.c*.

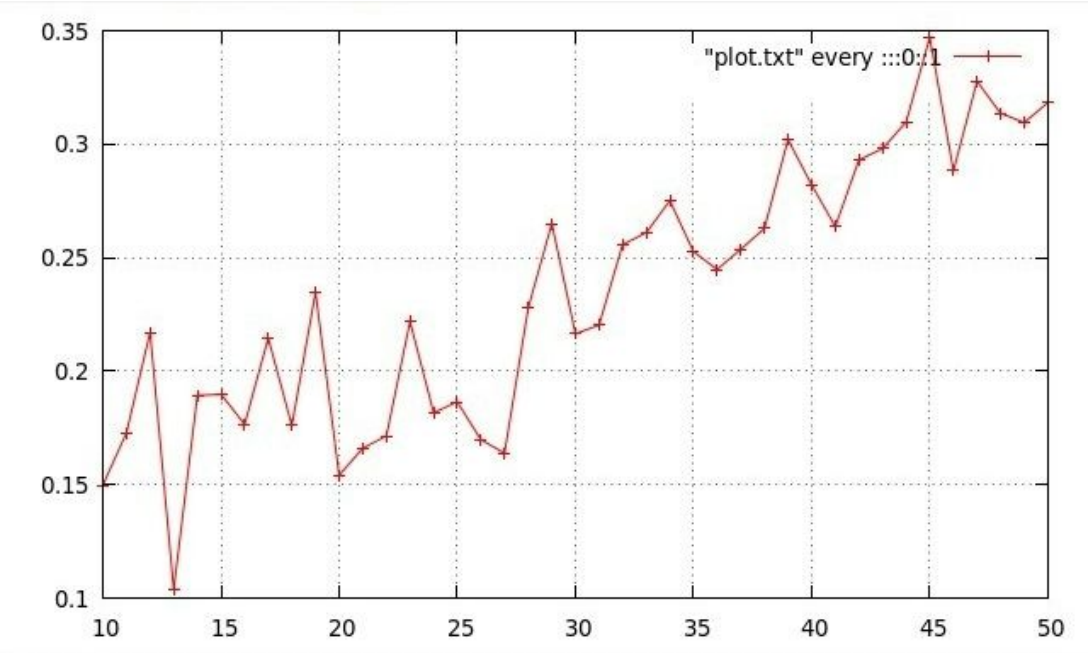
2. Αλγοριθμικές Επιλογές:

Για το clustering επιλέχθηκαν οι αλγόριθμοι που αναφέρθηκαν προηγουμένως.

Για το διάνυσμα αποστάσεων r, επιλέγονται sqrt(r)*(sqrt(r)-1)/2 (~=r) σημεία τυχαία από τα 2 μόρια.

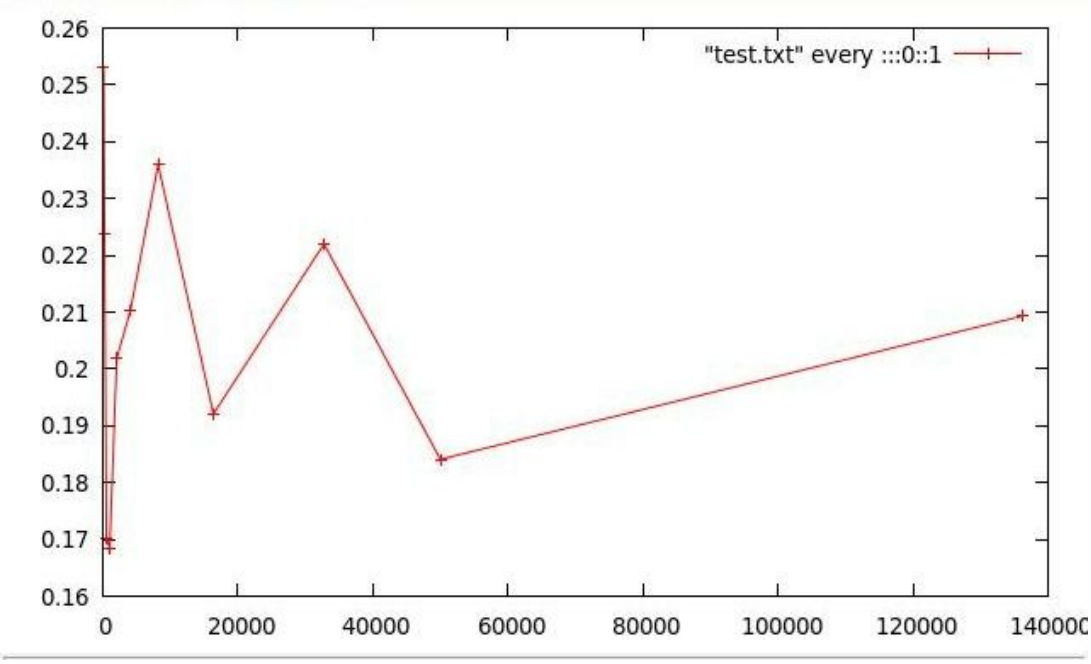
3. Αποτελέσματα εκτελέσεων - Validation:

Η εκτέλεση της εφαρμογής για την cRMSD μας έδωσε optimal k value k=19, με τιμή silhouette = 0.234. Παρακάτω φαίνεται η γραφική παράσταση των 2 μεγεθών:



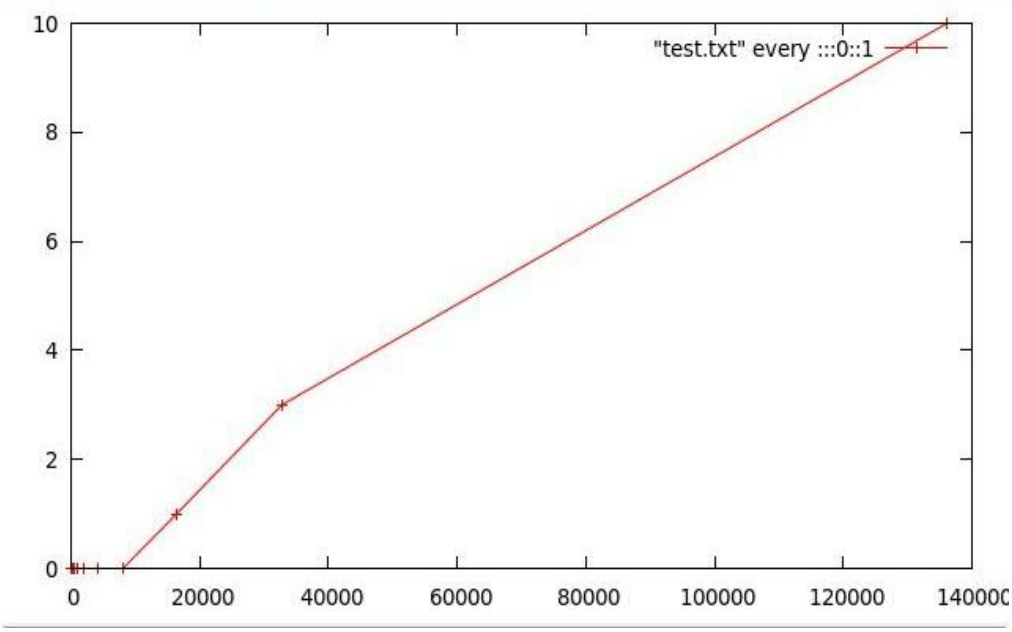
Από την γραφική παράσταση αυτή φαίνεται πως για k=19 έχουμε μια αρκετά καλή τιμή Silhouette, χωρίς να επιβαρυνόμαστε με πολλαπλά clusters, τα οποία καθυστερούσαν αισθητά την εφαρμογή.

Συνεπώς, επιλέγοντας από την εφαρμογή του cRMSD το k=19, για το dRMSD παρατηρούμε τις παρακάτω σχέσεις:



Είναι λογικό ότι για μικρές τιμές του r, ο αλγόριθμος διαλέγει ουσιαστικά λίγες και τυχαίες αποστάσεις. Έτσι, το clustering είναι πολύ εκφυλισμένο και ο Silhouette έχει αρκετές διακυμάνσεις. Όσο όμως μεγαλώνει η τιμή του r, προχωράμε σε ένα πιο ουσιώδες clustering και η τιμή του Silhouette σταθεροποιείται.

Όσο για τον χρόνο εκτέλεσης, παρατηρούμε ότι είναι γραμμικός ως προς την αύξηση του r, όπως και φαίνεται παρακάτω:


"test.txt" every :::0::1
10
8
6
4
2
0
0
20000
40000
60000
80000
100000
120000
140000