ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών

ΤΜΗΜΑ
ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Project

Μέρος 2ο

Όνομα
Κυλάφη Χριστίνα-Θεανώ

ΑΜ
1115201200077

Αθήνα, 2018

Η εργασία έχει υλοποιηθεί σε **γλώσσα C**, στο πρόγραμμα "**sublime text**" ενώ παράλληλα δοκιμαζόταν σε περιβάλλον **linux** σε Virtual Machine(**VirtualBox**) και γινόταν υποβολή (**push**) με τη χρήση εντολών **git** στο **bitbucket**, σε **private repository**.

Στο φάκελο, περιλαμβάνονται τα εξής αρχεία εκτός από το **readme** :
makefile, **lsh.c**, **lsh.h**, **cube.c**, **cube.h, cubefuns.c**, **cubefuns.h**, **hash.c**, **hash.h**, **structfuns.c**, **structs.h**, **extrafuns.c**, **extrafuns.h**, **cluster.c**, **cluster.h**, **steps_of_clustering.c**, **steps_of_clustering.h**, **initialization.c**, **initialization.h**, **assignment.c**, **assignment.h**, **update.c**, **update.h**, **cluster.conf**, **output**, **readme_part1**(το readme του πρώτου μέρους της εργασίας, που αφορά τους 2 αλγορίθμους - LSH/hyperCUBE - που αξιοποιούμε στο part 2 για assignment των points στα clusters), **αρχεία** συγκριτικών μετρήσεων μεταξύ των 12 αλγορίθμων(μορφής **results_<date>_<time>.txt**). **Δεν** έχω συμπεριλάβει το αρχείο **dataset**(**.csv**) λόγω μεγέθους, όμως χρειάζεται να βρίσκεται στον ίδιο φάκελο για να εκτελεστεί σωστά το προγραμμα.

Με την εντολή "**make**" παράγεται το εκτελέσιμο "**cluster**".

Η σύνταξη για την εκτέλεση του παραπάνω, είναι:
./cluster –i <input file> **–c** <configuration file> **-o** <output file> **-d** <metric> **-complete**(optional)

(Έχω προσθέσει **ελέγχους** για μη ορθή σύνταξη - εαν δοθούν λάθος - εκτός ορίων τιμές, το πρόγραμμα εκτελείται κανονικά, με **default** τιμές - και δυνατότητα **επανεκτέλεσης** του εκάστοτε αλγορίθμου, μέχρι ο χρήστης να πληκτρολογήσει " **exit** ").

> Λίγες λεπτομέρειες για τα αρχεία:
—cluster.c: Η βασική συνάρτηση που ξεκινά το επόμενο "επίπεδο" βασικών συναρτήσεων.
—cluster.h: Όπως αναφέρεται και παρακάτω, εδώ γίνονται define κάποιες παράμετροι, πολλαπλασιαστές, όρια τιμών που αφορούν συνθήκες τερματισμού, και includes των επικεφαλίδων που χρειάζονται.
—steps_of_clustering.c - .h: Εδώ βήμα βήμα, γίνεται η αρχικοποίηση και επαναληπτικά η ανάθεση points στα clusters και το update των κεντροειδών, και τέλος το silhouette(evaluation step) και εκτύπωση στο αρχείο. Ανάλογα με τις συνθήκες τερματισμού που ελέγχονται σε κάθε επανάληψη μετά το update, τερματίζει ή συνεχίζει η διαδικασία της επανάθεσης σημείων και επανενημέρωσης των κεντροειδών των clusters. Εδώ βρίσκεται και ο συνδυαστικός αλγόριθμος (12 συνδυασμοί), σε περίπτωση που δε δοθούν συγκεκριμένες επιλογές αλγορίθμων.
—initialization.c - .h: Οι αλγόριθμοι αρχικοποίησης των κεντροειδών.
—assignment.c -.h: Οι αλγόριθμοι ανάθεσης των σημείων στις συστάδες.
—update.c -.h: Οι αλγόριθμοι ενημέρωσης των κεντροειδών.
—lsh.c - .h: Καλεί διαδοχικά όλες τις απαραίτητες συναρτήσεις ώστε να δημιουργηθούν οι δομές και να εκτελεστεί ο αλγόριθμος **LSH**
—hash.c - hash.h: Τα structs / δομές/ συναρτήσεις που χτίζουν τον αλγόριθμο **LSH**.
—cube.c - .h: Αντίστοιχα με το lsh.c, το αρχείο αυτό, χειρίζεται τις συναρτήσεις για τη δημιουργία δομών και γενικά τις διαδικασίες ώστε να "έχει στο τέλος χτιστεί" βήμα βημα ο αλγόριθμος **hyperCUBE**
—cubefuns.c - cubefuns.h: Συναρτήσεις και δομές σχετικές με τον αλγόριθμο του υπερκύβου (δημιουργία, εκτέλεση αλγορίθμου, εκτύπωση κύβου για έλεγχο, κλπ)
—structfuns.c - structs.h: Οι συναρτήσεις και οι δομές που σχετίζονται με το χειρισμό των points και clusters.
—extrafuns.c: Βοηθητικές συναρτήσεις που αφορούν **διάβασμα** από αρχεία, **εκτυπώσεις** με συγκεκριμένη μορφή, **εγγραφή** σε αρχεία, **silhouette**, **input** από χρήστη, κλπ
—cluster.conf: Δεδομένα που χρειάζεται ο αλγόριθμος όπως πλήθος clusters, πλήθος hashTables για τον **LSH** όπως και πλήθος h functions(παραμέτρους όπως probes και points_to_check, για τον hyperCUBE , και άλλες όπως το w για την h(p) της ευκλείδειας μετρικής ή κάποιες μεταβλητές που χρησιμοποιούνται σαν πολλαπλασιαστές ή συνθήκες τερματισμού κλπ, γίνονται define στο **cluster.h** μαζί με τα includes των απαραίτητων επικεφαλίδων).

Το πρόγραμμα έχει ως εξής (**γενική ιδέα**) :

> Έχουν παραχθεί ένα εκτελέσιμο, το **cluster.**
CLUSTER: Αρχικοποιεί τα κεντροειδή με έναν από 2 τρόπους κάθε φορά.

> Παρακάτω αναλύονται οι **αλγόριθμοι** (υλοποίηση - συναρτήσεις - διαδικασίες) και παρουσιάζονται κάποιες ενδεικτικές εκτελέσεις.

> Κύριες δομές :

—> List of Input Points (struct multipoints*) στην οποία **αποθηκεύονται όλα τα points** που διαβάζουμε από το file που έχει επιλέξει ο χρήστης. Το διάβασμα και η αποθήκευσή τους, γίνεται από τη συνάρτηση **save_points( )** (στο αρχείο **extrafuns.c**), η οποία αποθηκεύει τις συντεταγμένες (μία μία με τη συνάρτηση **multipoint_insertcoord( )**) του κάθε σημείου στον αντίστοιχο κόμβο (κάθε σημείο το εισάγει στη λίστα με την **multipoint_insertpoint( )**), το όνομά του και τον επόμενό του (σε προηγούμενο στάδιο υλοποίησης, είχα και ένα id κάθε σημείου- σε περίπτωση που έπρεπε να το δημιουργήσω εγώ εσωτερικά στο πρόγραμμα).

—> Array of Clusters (struct cluster*) που αποθηκεύει τα **κεντροειδή** των συστάδων καθώς και τα **points που έχουν ανατεθεί** σε κάθε μία συστάδα τόσο στην αρχικοποίηση όσο και στα βήματα της ανάθεσης και της ενημέρωσης. Κρατάει τις απαραίτητες πληροφορίες όπως το πλήθος των totpoints_in_cluster στο κάθε cluster, κλπ.

—> List of points_in_cluster (struct points_in_cluster*) είναι ακριβώς η λίστα των σημείων που έχουν ανατεθεί στο κάθε cluster. Κρατάω και τον τελευταίο κόμβο της λίστας στη δομή "**cluster"** για λόγους διαχείρισης της δομής αργότερα.

—> Πίνακας αποστάσεων διάστασης ~(points_in_cluster^2) / 2 (τριγωνικός πίνακας) όπου κρατώνται οι αποστάσεις μεταξύ των σημείων του κάθε cluster, για λόγους χρονικής βελτιστοποίησης του αλγορίθμου ενημέρωσης κεντροειδών **PAM**, ώστε ν' αποφευχθεί η πολυπλοκότητα **O(** points_in_cluster ^ 2 **)**.

—> Array of Silhouette values οι τιμές του **silhouette** για κάθε cluster καθώς και ο **μέσος όρος** τους.

—> Array of results (struct results* array[ ]), η οποία δομή, κρατάει τα **αποτελέσματα**(χρόνους και τιμές silhouette) από τους **12 συνδυαστικούς αλγορίθμους** (όταν δεν επιλέγει συγκεκριμένους ο χρήστης), ώστε να τυπωθούν και να **μελετηθούν** μετά το πέρας της διαδικασίας.

> Βασικές Συναρτήσεις - Αλγόριθμοι :

—> save_points( ) : Διαβάζει από το input αρχείο που επέλεξε ο χρήστης, τα input points, και τα αποθηκεύει στη δομή **struct multipoints* Mpointlist**.

—> initialization functions : Συναρτήσεις αρχικοποίησης των κεντροειδών των k συστάδων που θα δημιουργήσουμε.

- Η συνάρτηση **k_means_plus_init( )** μαζί με ένα μικρό σύνολο από βοηθητικές συναρτήσεις για την υλοποίησή της, ακολουθεί τον αλγόριθμο **k-means++**. Δηλαδή, θέτει το **πρώτο** κεντροειδές με **τυχαίο** τρόπο (ομοιόμορφη επιλογή από τα n σημεία του dataset). Έπειτα, για την (πιθανοτική) επιλογή κάθε επόμενου, δημιουργεί τα partial sums του τετραγώνου της ελάχιστης απόστασης μεταξύ των σημείων και κάποιου κεντροειδούς (μέχρι τα τότε ορισμένα κεντροειδή) , επιλέγει έναν τυχαίο double στο διάστημα [0 … partial_sum[n]] και στη συνέχεια, διατρέχει τον πίνακα με τα partial sums, ψάχνοντας τη θέση (position) στην οποία το partial sum, έχει την ιδιότητα “ (double) x >= partial_sum[position] ”. Το position-οστό σημείο της δομής MpointsList (στην οποία έχουν αποθηκευτεί όλα τα σημεία) θα αποτελέσει το νέο κεντροειδές.
- Η **simple_rand_init( )**, αρχικοποιεί τα κεντροειδή τυχαία, δηλαδή διαλέγει από το πλήθος των σημείων του inputfile, k τυχαία, ώστε να τα θέσει σαν αρχικά κεντροειδή.

—> assignment functions : Στο βήμα αυτό, έχω είτε από το βήμα της αρχικοποίησης, είτε από εκείνο της ενημέρωσης τα k κεντροειδή και το πρώτο point(αναθέτω και το κεντροειδές πριν την αρχή της διαδικασίας ανάθεσης, ώστε στους μέσους όρους και) και αναθέτω με τους παρακάτω αλγορίθμους όλα τα σημεία σε ένα από αυτά τα clusters (με τα αντίστοιχα κεντροειδή).

- **Lloyd’s :** Πολύ απλή ανάθεση κάθε σημείου στη συστάδα της οποίας το κεντροειδές είναι **πιο κοντά** σε αυτό το σημείο, **ελέγχοντας** κάθε φορά να μην είναι το ίδιο το σημείο ένα από τα κεντροειδή και βρίσκοντας έπειτα με τη συνάρτηση **closest_centroid_to_point( )** το κοντινότερο κεντροειδές για ν’ αναθέσουμε το σημείο στο αντίστοιχο cluster.
- **Range Search (LSH / hyperCUBE) :** Στο πρώτο μέρος, είχα υλοποιήσει μια απλή λίστα με τα σημεία που βρίσκονται εντός της ακτίνας “**radius**” που κάθε φορά επέλεγε ο χρήστης. Με βάση λοιπόν αυτό, κάνω **range search** για κάθε **κεντροειδές** που ήδη έχει επιλεγεί από το initialization step ή το update step, εκτελώ πλήρως τον αλγόριθμο **LSH** ή **hyperCUBE** αντίστοιχα(αφού δημιουργήσω στην αρχή -μία φορά- την αντίστοιχη δομή για ολόκληρο τον αλγόριθμο), βρίσκω τους **γείτονες** εντός της ακτίνας κάθε φορά (εδώ έχω ένα αρχικό range το οποίο σε κάθε υπο-βήμα του assignment διπλασιάζεται) και τελικά έχω παράξει τη λίστα των **μοναδικών** γειτόνων του συγκεκριμένου κεντροειδούς. Με τις συναρτήσεις **point_assignment_cube( )** και **point_assignment_lsh( )** αναθέτω αυτούς τους γείτονες στο cluster του κεντροειδούς για το οποίο έκανα την αναζήτηση, συνεχίζω για όλα τα κεντροειδή **εντός του ίδιου range**, και αφού εκτελέσω την αναζήτηση γειτόνων για όλα τα κεντροειδή, με τη συνάρτηση **lock_points( )**, κλειδώνω την ανάθεση των points μέχρι εκείνη τη στιγμή. Συγκεκριμένα, έχω χρησιμοποιήσει 2 μεταβλητές στη δομή multipoints, τις **centroid** που αφορά τη συστάδα στην οποία μέχρι τότε ανήκει το point και τη **locked**. Με την προαναφερθείσα συνάρτηση λοιπόν, θέτω τη μεταβλητή **locked = 1**, ώστε στο επόμενο range , εαν κάποιο point βρεθεί στην ακτίνα κάποιου άλλου κεντροειδούς, να μην αλλάξει συστάδα. Στο ίδιο range, εαν έχουμε collisions, συγκρίνουμε τις αποστάσεις μεταξύ του σημείου και των κεντροειδών που ήρθαν σε σύγκρουση. Έχω θέσει 2 διαφορετικά κριτήρια τερματισμού του διπλασιασμού του range και επανεκτέλεσης του αλγορίθμου, είτε το πλήθος των clusters που δεν τους ανατίθεται κάποιο point να ξεπερνουν τα μισά, είτε να έχουμε ξεπεράσει κάποιο συγκεκριμένο πλήθος φορών διπλασιασμού του range. Όταν λοιπόν σταματήσει η επαναληπτική εκτέλεση της διαδικασίας, εκτελείται η συνάρτηση **assign_points_left( )**, η οποία αναθέτει τα υπολοιπόμενα σημεία, με τον ίδιο τρόπο που γινόταν και στον αλγόριθμο Lloyd’s (στο cluster με το κοντινότερο κεντροειδές στο σημείο).

—> update functions : Έχοντας αναθέσει κάθε σημείο σε ένα από τα k clusters που έχουν δημιουργηθεί, ενημερώνω τα κεντροειδή με τους παρακάτω αλγορίθμους.

- **K-means** : Σε κάθε cluster, αθροίζω τις συντεταγμένες των σημείων του, διαιρώ με το πλήθος των points του, και έτσι βρίσκω το **κέντρο** της συστάδας, το οποίο θ' αποτελέσει το νέο, ενημερωμένο κεντροειδές του.
- **PAM** (improved like Lloyd's) : Εδώ, έγινε χρήση τριγωνικού πίνακα αποστάσεων για να επιτευχθεί χρονική κυρίως βελτίωση του προγράμματος. Αυτός ο αλγόριθμος, ανανεώνει τα κεντροειδή, αντικαθιστώντας κάθε φορά το προηγούμενο κέντρο του cluster, με το σημείο εκείνο του cluster, στο οποίο **ελαχιστοποιείται** το **άθροισμα** των **αποστάσεων** του νέου κέντρου από όλα τα σημεία της συστάδας. Έχοντας κρατήσει τις αποστάσεις του κάθε σημείου από όλα τα υπόλοιπα σε έναν πίνακα, θα έχω και πολύ λιγότερο χρόνο εύρεσης της απόστασης. Έτσι, για κάθε cluster, για κάθε point, υπολογίζεται το άθροισμα αυτό, και το point που το **ελαχιστοποιεί**, σε κάθε cluster, τίθεται ως το νέο κεντροειδές αυτού.

Έπειτα από την εκτέλεση του βήματος με έναν από τους δύο αλγορίθμους, όπως αναφέρεται παρακάτω, ελέγχεται (με τον αντίστοιχο τύπο) η **objective function** των νέων κεντροειδών και το **ποσοστό της αλλαγής** της, ώστε σε συνδυασμό με κάποιες άλλες συνθήκες τερματισμού, ν' αποφασιστεί **αν θα τερματίσει** η επανάληψη reassignment-update, ή όχι.

—> steps_of_clustering( ) : Εδώ, ανάλογα με την απάντηση του χρήστη, εκτελείται είτε ένας αλγόριθμος που σχηματίζεται από το σύνολο των επιλογών του χρήστη (**users_choice( )**) για κάθε βήμα (**initialization**, **assignment**, **update**), είτε εκτελούνται και οι 12 αυτοί συνδυασμοί (**all_combinations( )**). Αφού γίνει η αρχικοποίηση των κεντροειδών, εκτελούνται σε επανάληψη τα βήματα assignment / reassignment και update. Πριν την έναρξη κάθε reassignment, ελέγχεται η αντίστοιχη objective function του κάθε αλγορίθμου ενημέρωσης και συγκρίνεται με την προηγούμενη, ώστε αν η βελτίωσή της είναι κάτω από 1%, να σταματάει η επανάληψη. Επίσης, 2 ακόμα συνθήκες τερματισμού είναι το πλήθος των επαναλήψεων reassignment-update καθώς και η περίπτωση που τα κέντρα μένουν ακριβώς τα ίδια.

—> struct functions : Συναρτήσεις για τη **διαχείριση** των **δομών** (struct multipoints, struct cluster), όπως , initialization, save, get, get_size, add, remove, print, delete, κλπ.

—> extra functions: Βοηθητικές συναρτήσεις σχετικά με την επιλογή των αλγορίθμων, την **αρχικοποίηση** των παραμέτρων του προγράμματος, **εισαγωγή** δεδομένων από το χρήστη, καθώς και συναρτήσεις για την **εξέταση** της αποτελεσματικότητας των αλγορίθμων που χρησιμοποιήθηκαν (evaluation - silhouette / χρόνοι).

—> print functions : Οικογένεια συναρτήσεων που εκτυπώνουν κάποιες **δομές** (για έλεγχο τιμών κατά τη διάρκεια της ανάπτυξης), **δεδομένα**, **σχεδιαστικά** στοιχεία, **αποτελέσματα** και αναλυτικά (με τη χρήση του argument -complete κατά την εκτέλεση του προγράμματος) ή μή, τις **συστάδες** που έχουν δημιουργηθεί με τα στατιστικά του κάθε αλγορίθμου.

—> delete functions : Οικογένεια συναρτήσεων για **απελευθέρωση μνήμης** (διαγραφή δομών που χρησιμοποιήθηκαν κατά τη διάρκεια εκτέλεσης του εκάστοτε αλγορίθμου).

> Παρατηρήσεις :

—> Έχει γίνει χρήση του πρώτου μέρους της εργασίας (**LSH-CUBE**). Τροποποιήθηκε κατάλληλα, ώστε να ικανοποιήσει τις απαιτήσεις του τωρινού μέρους, δηλαδή το **Range Search** στο **βήμα** της **ανάθεσης** στοιχείων στα clusters.

—> Έχει χρησιμοποιηθεί γενικά μεγάλος βαθμός **κατακερματισμού** των συναρτήσεων στο πρόγραμμα, ώστε να είναι προσαρμόσιμο και εύκολα συντηρήσιμο. Κάθε κύρια συνάρτηση, "σπάει" σε μικρότερες και πιο βασικές, για την εύκολη **επαναχρησιμοποίηση** και διόρθωση/ **έλεγχό** τους.

—> Έχουμε **δύο μετρικές** και πάλι (επηρεάζουν μόνο τη συνάρτηση της απόστασης **get_dist_multipoints( )**, η οποία επιλέγει τον κατάλληλο τύπο της απόστασης που θα χρησιμοποιήσει για τα σημεία που θέλουμε να μετρήσουμε την απόσταση).

—> Σε κάθε εκτέλεση και των **12 συνδυασμών** (όταν γίνεται η ερώτηση για την επιλογή των αλγορίθμων, η επιλογή "**n**" - **no**), παράγεται κάθε φορά ένα αρχείο με όνομα της μορφής "**results_<date>_<time>.txt**" το οποίο περιέχει έναν πίνακα με το **silhouette** (μέσο όρο των μέσων όρων) καθώς και το **χρόνο εκτέλεσης** του κάθε αλγορίθμου και καταγράφει και τις παραμέτρους με τις οποίες έτρεξε το πρόγραμμα, για ανάλυση και μελέτη της απόδοσης των αλγορίθμων συγκριτικά.

—> Σε όλα τα βήματα, γίνονται οι απαραίτητοι **έλεγχοι** ώστε να μην υπάρχουν >= 2 ίδια κεντροειδή(θα υπήρχαν διπλότυπα clusters).

—> Τα **κενά clusters παραμένουν κενά**, ώστε αυτό να συμβάλλει στη σωστή αξιολόγηση του πλήθους των cluster όσο και των υπόλοιπων επιλογών και παραμέτρων που έχουν χρησιμοποιηθεί για την εκάστοτε συσταδοποίηση.

> Συμπεράσματα (result files) :

Παρατηρώντας τα αρχεία που έχουν παραχθεί από την εκτέλεση των 12 συνδυαστικών αλγορίθμων, καταλήγουμε στα παρακάτω **γενικά συμπεράσματα** / **παρατηρήσεις**:

- Οι αλγόριθμοι **επηρεάζονται** από πολλές **παραμέτρους** (μετρική, πλήθος clusters, παράμετροι lsh/cube, σύστημα εκτέλεσης του προγράμματος, επαναλήψεις εκτέλεσης, αρχικοποίηση κεντροειδών, κλπ) , διαφορετικές ο καθένας, με αποτέλεσμα να μη μπορεί να γίνει αντικειμενική σύγκριση μεταξύ τους, συνεπώς θα περιοριστούμε σε γενικότερη παρατήρησή τους, με βάση το **χρόνο εκτέλεσής τους** και την αξιολόγηση **silhouette**, με τις ίδιες παραμέτρους κάθε φορά.

- Ο αλγόριθμος **k-means++** της **αρχικοποίησης**, προσφέρει γενικά καλές τιμές silhouette (ανάλογα με τις περιπτώσεις).

- Στο βήμα του **assignment**, ο αλγόριθμος **Lloyd's** δίνει **πολύ καλά αποτελέσματα** (αναμενόμενο, αφού το κάθε point γίνεται assigned στο κοντινότερο κέντρο) και μάλιστα με **όχι τόσο μεγάλες τιμές χρόνου εκτέλεσης**(φαίνεται να είναι ο πιο κατάλληλος με καλή σχέση χρόνου - αποτελεσματικότητας).

- Ο αλγόριθμος **LSH** στο **assignment** σε συνδυασμό με τον **k-means** στην **ενημέρωση** και λίγες επαναλήψεις, δε δίνει καθόλου καλά αποτελέσματα (ειδικά στις εκτελέσεις με **λίγα clusters** - **10/15/20/30** ακόμα και στα **50**, ενώ με **ενημέρωση PAM**, δίνει λίγο καλύτερα αποτελέσματα σε κάποιες περιπτώσεις, και σίγουρα αυξάνοντας τις επαναλήψεις.

- Παρατηρείται ότι (ειδικά με **πολλά clusters**), χρειάζονται **πολλές επαναλήψεις** (> 15) και ίσως κατάλληλος συνδυασμός πλήθους **h functions** - **hashtables**, ώστε να παραχθούν "**καλά**" αποτελέσματα από τον **LSH**(με **6 h functions** και **7 hashtables**, παρατηρούνται καλές τιμές στην cosine μετρική) και **CUBE**(με **4 h functions** και **8 hashtables**, παρατηρούνται **καλύτερες** τιμές - στην **cosine** μετρική), αλλιώς το **silhouette** παραμένει κοντά στο **0** (πολλές φορές και **< 0** στον **LSH**). Στην περίπτωση των **50** clusters, είχαμε πολύ **καλύτερα αποτελέσματα** στον αλγόριθμο **CUBE** σε σχέση με τον **LSH** και αντίθετα, **μεγαλύτερους χρόνους** εκτέλεσης στον **CUBE**, από εκείνους του **LSH** στην περίπτωση με τις **4 h functions - 8 hashtables**, ενώ είχαμε αντίθετα αποτελέσματα στην περίπτωση των **6 functions - 7 hashtables**(καλύτερο **LSH** - όχι τόσο καλό **CUBE**).

- Ο αλγόριθμος **ενημέρωσης PAM**, σύμφωνα με τιμές από κάποιες εκτελέσεις που παρατηρούμε στους συνδυασμούς 1/2 x 1/2/3 x **2**, είχε **μεγαλύτερους χρόνους εκτέλεσης** από τον **k-means update**, χωρίς απαραίτητα να προσφέρει καλύτερα αποτελέσματα **silhouette**.

- Στην εκτέλεση με τα **50 clusters**(στην cosine μετρική), είχαμε πολύ καλά αποτελέσματα silhouette, με **λίγες επαναλήψεις**, στους αλγορίθμους **1/2 x 1 x 1** και **1/2 x 1 x 2** (**Lloyd's assignment + k-means update**), και ειδικά στους συνδυασμούς **1/2 x 1 x 2**, παρατηρούνται οι μικρότεροι χρόνοι (**Lloyd's assignment + PAM update** - αναμενόμενα τα καλύτερα **και** χρονικά, αφού θα έχουν πολύ γρήγορη σύγκλιση της objective function ή ακόμα και μη αλλαγή των κεντροειδών, σε μικρό πλήθος επαναλήψεων). Στην **ευκλείδεια μετρική**, δεν παρατηρήθηκε το ίδιο.

- Γενικότερα, παρατηρήθηκαν αξιοσημείωτες διαφορές στην απόδοση των αλγορίθμων όταν άλλαζαν οι μετρικές (**cosine** - **euclidean**), καθώς και οι τιμές των **clusters**, **h functions**, **hashtables**, **w**, **probes**, **points_to_check**, **επαναλήψεις**, **range limit**.

- Επίσης, στην εκτέλεση των **50** και **100 clusters**, είδαμε κάποια **βελτίωση** των τιμών του **silhouette**, σε σχέση με τα λιγότερα **clusters**(**< 50**) που είχαν προηγούμενως δοκιμαστεί.