{ Ελευθερία Εκατομμάτη sdi2000048, Θεοδώρα Αρχοντάκη sdi2000014 }

LSH:

Random.h, Random.cpp:

Τα δύο αυτά αρχεία έχουν παραμείνει ίδια με την προηγούμενη υλοποίηση.

Image_vector.h:

Έχουμε τον ορισμό μίας νέας συνάρτησης: «create_image_initial». Και έχουμε ορίσει ως extern δύο ακεραίους Dim, Dim_initial ο πρώτος είναι η διάσταση στο μικρότερο χώρο και ο δεύτερος η διάσταση στον αρχικό χώρο δηλαδή το 784. Ακόμα έχουμε πλέον τέσσερις πίνακες στους οποίους αποθηκεύουμε τα διανύσματα των εικόνων. Το Images που περιέχει τα διανύσματα των εικόνων στον μικρότερο χώρο και το Images_initial για τα διανύσματα των εικόνων στον αρχικό χώρο και υπάρχουν αντίστοιχοι πίνακες και για το query αρχείο.

Create_image_vector.cpp:

Η «create_image» έχει αλλάξει και αυτό που κάνει είναι να διαβάζει από το αρχείο εισόδου τα διανύσματα των εικόνων και τα αποθηκεύει στον κατάλληλο πίνακα. Επίσης αρχικοποιεί και την τιμή της μεταβλητής Dim η οποία θα είναι ίση με τον αριθμό των στηλών στο αρχείο εισόδου.

Η «create_image_initial» είναι ίδια με την υλοποίηση που είχαμε πριν και αποθηκεύει σε έναν vector τύπου MNIST_Image τα διανύσματα των εικόνων του αρχικού χώρου. Η υλοποίηση των υπόλοιπων συναρτήσεων είναι ίδια.

Hash.h:

Έχουμε προσθέσει τον ορισμό μίας νέας συνάρτησης «Euclidian_dist_initial».

Hash.cpp:

Οι συναρτήσεις: «Hash initialize», «HashFunctions_initialize», «Euclidian Hash», «Hash clear», «Query Hash» έχουν την ίδια υλοποίηση.

Η συνάρτηση «Euclidian_dist» υπολογίζει πλέον την απόσταση μεταξύ δύο διανυσμάτων της μικρότερης διάστασης. Ενώ η «Euclidian_dist_initial» υπολογίζει την απόσταση μεταξύ δύο διανυσμάτων της αρχικής διάστασης.

Οπότε στις συναρτήσεις που εκτελούμε τον exhaustive αλγόριθμο «Euclidian_Nearest_NN», «Euclidian_NN» για την εύρεση του κοντινότερου γείτονα και του N-οστού κοντινότερου γείτονα καλούμε την συνάρτηση υπολογισμού της απόστασης για τα διανύσματα του αρχικού χώρου. Στις συνάρτησης για την εκτέλεση του LSH «LSH_Nearest_N», «LSH_RangeSearch», «N_Nearestneighbors_LSH» καλούμε την συνάρτηση για τον υπολογισμό της απόστασης στον χώρο της μικρότερης διάστασης.

lsh.cpp:

Είναι η main όπου αρχικά καλούμε την συνάρτηση Input για να διαβάσουμε τα ορίσματα και μετά το πρόγραμμα ζητάει από τον χρήστη να του δώσει τα αρχεία τύπου MNIST που περιέχουν τις εικόνες στην αρχική τους διάσταση και μετά εκτελεί τους αλγορίθμους. Αφού

εκτελέσει τους αλγορίθμους στη μειωμένη διάσταση καλεί το πρόγραμμα (με την συνάρτηση system) που εκτελεί τους αλγορίθμους στην αρχική διάσταση (784).

HYPERCUBE:

Image.h, Image.cpp:

Έχουμε κάνει τις ίδιες αλλαγές όπως και στον LSH. Έχουμε πάλι τέσσερις διαφορετικούς vectors στους οποίους αποθηκεύουμε τις εικόνες στην αρχική διάσταση και στην μικρότερη διάσταση.

Euclidian.h:

Έχουμε απλά προσθέσει την συνάρτηση «Euclidian_dist_initial» που υπολογίζει την απόσταση μεταξύ δύο διανυσμάτων διάστασης 784.

Euclidian.cpp:

Η συνάρτηση «Euclidian_dist_initial» υπολογίζει την απόσταση μεταξύ διανυσμάτων της αρχικής διάστασης ενώ η «Euclidian_dist» υπολογίζει πλέον την απόσταση διανυσμάτων της μικρότερης διάστασης.

Και για τις συναρτήσεις που εκτελούν τον exhaustive αλγόριθμο «Euclidian_Nearest_N», «Euclidian_NNN» για τον υπολογισμό της απόστασης καλούν την συνάρτηση «Euclidian_dist_initial».

Οι συναρτήσεις που εκτελούν τον αλγόριθμο του υπερκύβου «Random_projection_to_HyperCube», «Random_projection_to_HyperCube_N», «Random_projection_to_HyperCube_RangeSearch» καλούν την συνάρτηση της απόστασης του μικρότερου χώρου.

Cube.h, Cube_projection.cpp:

Δεν έχουμε αλλάξει κάτι στα συγκεκριμένα αρχεία απλώς στην εκτέλεση του αλγορίθμου του υπερκύβου καλείται πλέον η συνάρτηση υπολογισμού της απόστασης για την μικρότερη διάσταση.

cube.cpp:

Περιέχει την main. Αρχικά με την συνάρτηση Input διαβάζει τα δεδομένα που έχει δώσει ο χρήστης, μετά του ζητάει να δώσει τα αρχεία MNIST που περιέχουν τις εικόνες στην αρχική τους διάσταση και εκτελεί τους αλγορίθμους. Αφού εκτελέσει τους αλγορίθμους στη μειωμένη διάσταση καλεί το πρόγραμμα (με την συνάρτηση system) που εκτελεί τους αλγορίθμους στην αρχική διάσταση (784).

CLUSTER:

Image.h, Image.cpp:

Έχουμε κάνει τις ίδιες αλλαγές όπως και στον LSH όσον αφορά την συνάρτηση που διαβάζει τα αρχεία εισόδου και αποθηκεύει τα διανύσματα των εικόνων σε πίνακες. Η μόνη διαφορά είναι ότι εδώ έχουμε έναν vector<vector<int>> που αποθηκεύουμε τα διανύσματα των εικόνων στην μικρότερη διάσταση. Έχουμε και μια extern μεταβλητή Dim που είναι η διάσταση των εικόνων και είναι ίση με τον αριθμό των στηλών του αρχείου εισόδου.

Hash.h, Hash.cpp:

Η μόνη αλλαγή στα συγκεκριμένα αρχεία είναι η συνάρτηση της απόστασης "Euclidian_dist" που πλέον υπολογίζει την ευκλείδεια απόσταση σε διανύσματα διάστασης = latent_dimension. Και όπου χρειάζεται να υπολογιστεί η απόσταση καλείται αυτή η συνάρτηση.

Cluster.h, Cluster.cpp:

Στα συγκεκριμένα αρχεία δεν αλλάξαμε κάτι η υλοποίηση είναι ίδια με το παραδοτέο της πρώτης εργασίας.

main.cpp:

Περιέχει την main. Αρχικά με την συνάρτηση Input διαβάζει τα δεδομένα που έχει δώσει ο χρήστης και εκτελεί τον αλγόριθμο της συσταδοποιήσης με την μέθοδο που έχει ζητήσει ο χρήστης. Αφού εκτελέσει τον αλγόριθμο του clustering στη μειωμένη διάσταση το πρόγραμμα ζητάει από τον χρήστη να δώσει το αρχείο MNIST που περιέχει τις εικόνες στην αρχική τους διάσταση και (με την συνάρτηση system) εκτελεί τους αλγορίθμους στην αρχική διάσταση (784).

ΕΡΓΑΣΙΑ 2: GNNS, MRNG:

Random.h, Random.cpp, Graph.h, Graph.cpp, algorithms.h, algorithms.cpp:

Στα συγκεκριμένα αρχεία δεν έχουμε αλλάξει κάτι στην υλοποίηση.

Image.h, Image.cpp:

Έχουμε κάνει τις ίδιες αλλαγές όπως και στον LSH. Έχουμε πάλι τέσσερις διαφορετικούς vectors στους οποίους αποθηκεύουμε τις εικόνες στην αρχική διάσταση και στην μικρότερη διάσταση.

LSH.h, LSH.cpp:

Έχουμε την συνάρτηση «Euclidian_dist» που υπολογίζει την απόσταση δύο διανυσμάτων της μικρής διάστασης και την «Euclidian_dist_initial» που υπολογίζει την απόσταση δύο διανυσμάτων της αρχικής διάστασης. Οπότε στις συναρτήσεις που εκτελούν τους αλγορίθμους του LSH καλούμε την συνάρτηση της απόστασης της μικρής διάστασης και για τον exhaustive αλγόριθμο καλούμε την συνάρτηση που υπολογίζει την απόσταση για τη διάσταση διανυσμάτων 784.

main.cpp:

Περιέχει την main. Αρχικά με την συνάρτηση Input διαβάζει τα δεδομένα που έχει δώσει ο χρήστης, μετά του ζητάει να δώσει τα αρχεία MNIST που περιέχουν τις εικόνες στην αρχική τους διάσταση και εκτελεί τους αλγορίθμους. Αφού εκτελέσει τους αλγορίθμους στη μειωμένη διάσταση καλεί το πρόγραμμα (με την συνάρτηση system) που εκτελεί τους αλγορίθμους στην αρχική διάσταση (784).

Σε κάθε φάκελο LSH, Cube, Cluster, ergasia2, υπάρχει ένας ακόμα φάκελος (LSH_initial, Cube_initial, Cluster_initial, initial αντίστοιχα) που περιέχει τους αλγορίθμους για την αρχική διάσταση (784) και στην ουσία είναι τα παραδοτέα της πρώτης και της δεύτερης εργασίας. Τα αρχεία αυτά είναι ίδια με αυτά που έχουμε παραδώσει στις πρώτες δύο εργασιές.

Reduce.py:

Φόρτωση Δεδομένων MNIST:

- Χρησιμοποιείται η συνάρτηση "load_mnist_images" για τη φόρτωση των εικόνων από ένα αρχείο MNIST.
- Τα δεδομένα εκπαίδευσης (X_train) και ελέγχου (X_test) παίρνονται από το Keras MNIST dataset και κανονικοποιούνται.

Δημιουργία και Εκπαίδευση του Autoencoder:

- Ορίζονται οι συναρτήσεις "encoder" και "Autoencoder" που καθορίζουν το μοντέλο του autoencoder.
- Η συνάρτηση "Neural_network" δημιουργεί και εκπαιδεύει το μοντέλο του autoencoder.

Αποθήκευση των Latent Vectors:

- Η συνάρτηση "save_latent_vectors" χρησιμοποιείται για την αποθήκευση των latent vectors σε ένα αρχείο κειμένου.

Καθορισμός Επιλογών Χρήστη:

- Η συνάρτηση "main" ρωτά τον χρήστη αν θέλει να δημιουργήσει ένα νέο μοντέλο ή να φορτώσει ένα υπάρχον.
- Στη συνέχεια, διαβάζει τα ονόματα των αρχείων από το command line.

Επεξεργασία Δεδομένων Εισόδου:

- Η συνάρτηση "main" διαβάζει τις παραμέτρους από το command line και εκτελεί τις αντίστοιχες ενέργειες.
- Φορτώνει τα δεδομένα εισόδου (dataset και query), κανονικοποιεί τις εικόνες και χρησιμοποιεί τον encoder για να αποθηκεύσει τα latent vectors.

Το πρόγραμμα θα ρωτήσει τον χρήστη αν θέλει να δημιουργήσει από την αρχή το νευρωνικό δίκτυο ή αν θέλει να χρησιμοποιήσει ένα ήδη υπάρχον το οποίο είναι αποθηκευμένο με το όνομα «autoencoder.h5».

Οδηγίες μεταγλώττισης:

- Για τον αλγόριθμο LSH στην μειωμένη διάσταση. Βρισκόμαστε στον φάκελο LSH και καλούμε: make lsh έτσι δημιουργείται το εκτελέσιμο αρχείο lsh. Πρέπει να πάμε και στον φάκελο LSH_initital να κάνουμε make για να δημιουργηθεί το εκτελέσιμο lsh για την αρχική διάσταση το οποίο θα καλέσουμε μέσα από την main του άλλου προγράμματος. Μπορούμε να το εκτελέσουμε με τους παρακάτω δύο τρόπους:
  - ./lsh σε αυτήν την περίπτωση το πρόγραμμα θα ζητήσει από τον χρήστη τις διαδρομές για τα κατάλληλα αρχεία και τις τιμές για τα k,L,N,R.
  - ./lsh –d <input_file> –q <query_file> -k <int> -L <int> -o <output_file> -N <int> -R <radius>

  Στην πρώτη περίπτωση αν ο χρήστης δεν θέλει να δώσει κάποια τιμή σε μία μεταβλητή τότε απλά γράφει "-" και σε αυτήν την περίπτωση χρησιμοποιούνται οι default τιμές. Με αυτόν τον τρόπο εκτελούνται οι αλγόριθμοι και για τις δύο διαστάσεις.

- Για τον αλγόριθμο προβολής στον υπερκύβο στην μειωμένη διάσταση. Βρισκόμαστε στον φάκελο Cube και καλούμε: make cube έτσι δημιουργείται το εκτελέσιμο αρχείο cube. Πρέπει να πάμε και στον φάκελο Cube_initital να κάνουμε make για να δημιουργηθεί το εκτελέσιμο cube για την αρχική διάσταση το οποίο θα καλέσουμε μέσα από την main του άλλου προγράμματος. Μπορούμε να το εκτελέσουμε με τους παρακάτω δύο τρόπους:
  - ./cube σε αυτήν την περίπτωση το πρόγραμμα θα ζητήσει από τον χρήστη τις διαδρομές για τα κατάλληλα αρχεία και τις τιμές για τα k,M,N,R, probes.
  - ./cube –d <input_file> –q <query_file> –k <int> -M <int> -probes <int> -o <output_file> -N <int> -R <int>

  Στην πρώτη περίπτωση αν ο χρήστης δεν θέλει να δώσει κάποια τιμή σε μία μεταβλητή τότε απλά γράφει "-" και σε αυτήν την περίπτωση χρησιμοποιούνται οι default τιμές. Με αυτόν τον τρόπο εκτελούνται οι αλγόριθμοι και για τις δύο διαστάσεις.

- Για τον αλγόριθμο GNNS, καλούμε: make graph_search έτσι δημιουργείται το εκτελέσιμο αρχείο graph_search για την μειωμένη διάσταση. Πρέπει να πάμε και στον φάκελο initial να κάνουμε make για να δημιουργηθεί το εκτελέσιμο graph_search για την αρχική διάσταση το οποίο θα καλέσουμε μέσα από την main του άλλου προγράμματος. Μπορούμε να το εκτελέσουμε με τους παρακάτω δύο τρόπους:
  - ./graph_search σε αυτήν την περίπτωση το πρόγραμμα θα ζητήσει από τον χρήστη τις διαδρομές για τα κατάλληλα αρχεία και τις τιμές για τα k, E, R, N όπως και την μέθοδο που θα εκτελέσει (1 για GNNS 2 για MRNG).
  - ./graph_search –d <input_file> –q <query_file> –k <int> -E <int> -R <int> -N <int> -m 1 -o <output_file>

  Στην πρώτη περίπτωση αν ο χρήστης δεν θέλει να δώσει κάποια τιμή σε μία μεταβλητή τότε απλά γράφει "-" και σε αυτήν την περίπτωση χρησιμοποιούνται οι default τιμές. Με αυτόν τον τρόπο εκτελούνται οι αλγόριθμοι και για τις δύο διαστάσεις.

- Για τον αλγόριθμο search on graph, καλούμε: make graph_search έτσι δημιουργείται το εκτελέσιμο αρχείο graph_search για την μειωμένη διάσταση. Πρέπει να πάμε και στον φάκελο initial να κάνουμε make για να δημιουργηθεί το εκτελέσιμο graph_search για την αρχική διάσταση το οποίο θα καλέσουμε μέσα από την main του άλλου προγράμματος. Μπορούμε να το εκτελέσουμε με τους παρακάτω δύο τρόπους:
    - ./graph_search σε αυτήν την περίπτωση το πρόγραμμα θα ζητήσει από τον χρήστη τις διαδρομές για τα κατάλληλα αρχεία και τις τιμές για τα N,L όπως και την μέθοδο που θα εκτελέσει (1 για GNNS 2 για MRNG).
    - ./graph_search –d <input_file> –q <query_file> –k <int> -E <int> -R <int> -N <int> -l <int> -m 2 -o <output_file>

    Στην πρώτη περίπτωση αν ο χρήστης δεν θέλει να δώσει κάποια τιμή σε μία μεταβλητή τότε απλά γράφει "-" και σε αυτήν την περίπτωση χρησιμοποιούνται οι default τιμές. Με αυτόν τον τρόπο εκτελούνται οι αλγόριθμοι και για τις δύο διαστάσεις.

- Για το reduce.py: python reduce.py -d <dataset> -q <queryset> -od <odataset.txt> -oq <oquery.txt>
    - Τα αρχεία dataset, queryset είναι αρχεία τύπου MNIST και τα odataset, oquery είναι αρχεία κειμένου.
- Για την συσταδοποίηση. Βρισκόμαστε στο φάκελο Cluster και καλούμε make cluster έτσι δημιουργείται το εκτελέσιμο αρχείο cluster για την μειωμένη διάσταση. Πρέπει να πάμε και στον φάκελο Cluster_initital να κάνουμε make για να δημιουργηθεί το εκτελέσιμο cluster για την αρχική διάσταση το οποίο θα καλέσουμε μέσα από την main του άλλου προγράμματος. Μπορούμε να το εκτελέσουμε με τον παρακάτω τρόπο:
    - $./cluster –i <input_file> –c <configuration_file> -o <output_file> -complete -m <method>

    Για το method έχουμε τρεις εναλλακτικές: Classic, LSH ή Hypercube. Και το «-complete» δεν είναι υποχρεωτικό να το δώσει ο χρήστης.

    Τα αρχεία input_file, query_file που χρησιμοποιούμε για είσοδο στις εκτελέσεις των προγραμμάτων LSH, Cube, Cluster, GNNS, MRNG πρέπει να είναι αρχεία .txt που έχουν γραμμές = αριθμό των εικόνων και στήλες = αριθμό του latent_vector και είναι τα αρχεία output που μας έδωσε η εκτέλεση του reduce.py.

Σημαντική παρατήρηση: επειδή σε όλες τις main των προγραμμάτων έχουμε την εντολή system που καλεί ένα άλλο πρόγραμμα τα linux βγάζουν ένα error για τα permission του αρχείου που πάμε να καλέσουμε μέσα στη main. Για να διορθωθεί αυτό το πρόβλημα στον κάθε φάκελο πρέπει να τρέξουμε τις παρακάτω εντολές:

- Στον φάκελο LSH: chmod +x ./LSH_initial/lsh
- Στον φάκελο Cube: chmod +x ./Cube_initial/cube
- Στον φάκελο Cluster: chmod +x ./Cluster_initial/cluster
- Στον φάκελο ergasia2: chmod +x ./initial/graph_search

Για τα αρχεία εξόδου:

Με την εκτέλεση των προγραμμάτων lsh, cube, cluster, graph_search στη μειωμένη διάσταση δημιουργούνται δύο αρχεία κειμένου που περιέχουν το output των αλγορίθμων. Το ένα έχει όνομα αυτό που του έχει δώσει ο χρήστης ("-o <outputfile>") και περιέχει τα αποτελέσματα από την εκτέλεση των αλγορίθμων στην μειωμένη διάσταση. Δημιουργείται και ένα ακόμα αρχείο .txt με όνομα "output_initial" και περιέχει τα αποτελέσματα από την εκτέλεση των αλγορίθμων στην αρχική διάσταση (784). Το κλάσμα AF εκτυπώνεται στο αρχείο εξόδου με όνομα αυτό που δίνει ο χρήστης.

Το κλάσμα AF: είναι

$$\frac{(\text{απόσταση προσεγγιστικού γειτονα από το query, στην μειωμένη διάσταση} - \text{αναγμένος στην αρχική διάσταση})}{(\text{απόσταση προσεγγιστικού γειτονα από το query, στην αρχική διάσταση})}$$

Ο προσεγγιστικός γείτονας είναι υπολογισμένος ή από LSH, Hypercube, GNNS, MRNG. Για τον προσεγγιστικό γείτονα στην μειωμένη διάσταση για να υπολογίσουμε την απόσταση του από το query τον ανάγουμε πρώτα στις 784 διαστάσεις.

Έχουμε και τα παρακάτω αρχεία:

Για τον φάκελο της εργασίας 2:

- Το **graph10.txt** περιέχει τον MRNG γράφο από το dataset με τις 10.000 εικόνες και N = 10.
- Το **graph5.txt** περιέχει τον MRNG γράφο από το dataset με τις 10.000 εικόνες και N = 4.
- Το **graph.txt** περιέχει τον MRNG γράφο από το dataset με τις 60.000 εικόνες και N = 5.

  Αν ο χρήστης θέλει να τρέξει το πρόγραμμα με έτοιμο γράφο τότε πρέπει απλά στο αντίστοιχο ".txt" αρχείο να δώσει το όνομα «graph.txt» και όταν το πρόγραμμα ρωτάει αν θέλει να χρησιμοποιήσει κάποιο έτοιμο γράφο να γράψει στο terminal "yes". Τότε το πρόγραμμα θα χρησιμοποιήσει τον έτοιμο γράφο που βρίσκεται στο αρχείο "graph.txt".

Σε όλους τους φακέλους έχουμε τα αρχεία output-dataset10.txt, output-query10.txt τα οποία είναι τα αρχεία output που επέστρεψε η εκτέλεση του reduce.py και χρησιμοποιούμε σαν input για την εκτέλεση των αλγορίθμων LSH, Cube, Clustering, GNNS, MRNG.

ΓΙΑ ΤΗΝ ΒΕΛΤΙΣΤΗ ΔΟΜΗ ΤΟΥ ΝΕΥΡΩΝΙΚΟΥ ΔΙΚΤΥΟΥ:

Χρησιμοποιήσαμε το εργαλείο optuna για την εύρεση των βέλτιστων υπερπαραμέτρων και καταλήξαμε στην παρακάτω δομή. Latent dimension = 10, batch size = 8, epochs = 27, αριθμός συνελικτικών στρωμάτων = 2, μέγεθος συνελικτικών φίλτρων 32, 64. Για την συγκεκριμένη δομή του νευρωνικού δικτύου το loss είναι πολύ χαμηλό της τάξης του 10^{-5}. Χωρίσαμε το dataset των εικόνων 70% training και 30% validation και δεν υπάρχει το φαινόμενο του overfitting.

ΣΧΟΛΙΑΣΜΟΣ ΑΛΓΟΡΙΘΜΩΝ:

- Για τον χρόνο παρατηρούμε ότι σε όλους τους αλγορίθμους στην χαμηλότερη διάσταση τρέχει αρκετά πιο γρήγορα σε σχέση με την αρχική (784). Ο χρόνος για τον exhaustive αλγόριθμο παραμένει ίδιος.
- Για το κλάσμα AF: είναι κοντά στο ένα (είτε κάτω από ένα είτε λίγο πάνω από αυτό). Με την ελάττωση της διάστασης υπάρχουν περιπτώσεις όπου βρίσκουμε κοντινότερο προσεγγιστικό γείτονα καλύτερο σε σχέση με την εκτέλεση των αλγορίθμων στις αρχικές διαστάσεις (784). Άρα με την ελάττωση της διάστασης μπορούν να συμβούν ένα από τα παρακάτω σενάρια:
    - Ο γείτονας που βρίσκουμε στην χαμηλότερη διάσταση να είναι πιο κοντά στο query σε σχέση με τον γείτονα που επιστρέφει η αναζήτηση στις 784 διαστάσεις (AF < 1). Άρα η ελάττωση των διαστάσεων επέφερε καλύτερο γείτονα.
    - Ο γείτονας που επιστρέφουν οι χαμηλότερες διαστάσεις να είναι σχεδόν το ίδιο κοντά με τον γείτονα των αρχικών διαστάσεων (AF ≈ 1).

ΣΧΟΛΙΑΣΜΟΣ ΣΥΣΤΑΔΟΠΟΙΗΣΗΣ:

Παρατηρούμε ότι όποια μέθοδο και αν χρησιμοποιήσουμε για να κάνουμε το clustering το silhouette βελτιώνεται σε σχέση με το silhouette που υπολογίζαμε στην αρχική διάσταση. Οπότε χρησιμοποιώντας το νευρωνικό δίκτυο η μειωμένη διάσταση βελτίωσε την αποδοτικότητα του αλγορίθμου. Παρατηρούμε ότι ο χρόνος έχει μειωθεί αισθητά, λογικό αφού ελαττώθηκε η διάσταση και άρα ο αλγόριθμος δουλεύει με μικρότερη ποσότητα δεδομένων.