

# Approximate Nearest Neighbor & Clustering

## Περιεχόμενα

<b>1</b>	<b>Μεταγλώττιση &amp; Εκτέλεση</b>	<b>3</b>
1.1	Μεταγλώττιση . . . . .	3
1.2	Εκτέλεση . . . . .	3
1.2.1	LSH . . . . .	3
1.2.2	Παράμετροι LSH . . . . .	3
1.2.3	Hyperspace . . . . .	3
1.2.4	Παράμετροι Hyperspace . . . . .	3
1.2.5	Cluster . . . . .	3
1.3	Input File . . . . .	3
1.4	Configuration File . . . . .	4
1.4.1	Πιθανές Παράμετροι . . . . .	4
<b>2</b>	<b>Εργαλεία Ανάπτυξης Project</b>	<b>4</b>
2.1	Version Control (Git/Github) . . . . .	4
2.2	Unit Testing (CppUnit) . . . . .	4
<b>3</b>	<b>Οργάνωση Αρχείων &amp; Φακέλων</b>	<b>5</b>
3.1	Οργάνωση Φακέλων . . . . .	5
3.1.1	Φάκελος bin . . . . .	5
3.1.2	Φάκελος data . . . . .	5
3.1.3	Φάκελος doc . . . . .	5
3.1.4	Φάκελος include . . . . .	5
3.1.5	Φάκελος obj . . . . .	5
3.1.6	Φάκελος src . . . . .	5
3.1.7	Φάκελος test . . . . .	5
3.2	Οργάνωση Αρχείων Κώδικα . . . . .	5
3.2.1	Αρχείο lsh.cpp . . . . .	5
3.2.2	Αρχείο cube.cpp . . . . .	5
3.2.3	Αρχείο cluster.cpp . . . . .	5
3.2.4	Αρχείο searcher.cpp/.hpp . . . . .	6
3.2.5	Αρχείο clusterCreator.cpp/.hpp . . . . .	6
3.2.6	Αρχείο hash_table.cpp/.hpp . . . . .	6
3.2.7	Αρχείο hasher.cpp/.hpp . . . . .	6
3.2.8	Αρχείο point.cpp/.hpp . . . . .	6
3.2.9	Αρχείο util.cpp/.hpp . . . . .	6
<b>4</b>	<b>Approximate Nearest Neighbor</b>	<b>6</b>
4.1	Latent Semantic Hashing (LSH) . . . . .	6
4.2	Υπερκύβος . . . . .	6

<b>5</b>	<b>Σύγκριση Μεθόδων LSH &amp; Hypercube</b>	<b>6</b>
5.1	Ενδεικτικές Εκτελέσεις . . . . .	6
5.1.1	LSH . . . . .	7
5.1.2	Hypercube . . . . .	7
5.1.3	Παρατηρήσεις & Σχόλια . . . . .	7
5.2	Κατανάλωση Χώρου . . . . .	8
5.2.1	LSH . . . . .	8
5.2.2	Υπερκύβος . . . . .	8
<b>6</b>	<b>Σύγκριση Μεθόδων Clustering</b>	<b>8</b>

# 1 Μεταγλώττιση & Εκτέλεση

## 1.1 Μεταγλώττιση

Τα προγράμματα μεταγλωττίζονται με τη χρήση του αρχείου Makefile και συγκεκριμένα τις εντολές:

- `make`
  - Για τη δημιουργία και όλων των εκτελέσιμων
- `make lsh`
  - Για τη δημιουργία μόνο του εκτελέσιμου `lsh`
- `make cube`
  - Για τη δημιουργία μόνο του εκτελέσιμου `cube`
- `make cluster`
  - Για τη δημιουργία μόνο του εκτελέσιμου `cluster`

## 1.2 Εκτέλεση

### 1.2.1 LSH

```
./lsh -d <input file> -q <query file> -k <int> -L <int> -o <output file>
```

### 1.2.2 Παράμετροι LSH

- $L$  είναι το πλήθος των hash tables που θα δημιουργηθούν
- $k$  είναι το πλήθος των συναρτήσεων κατακερματισμού  $h_i$  ανά hash table

### 1.2.3 Hypercube

```
./cube -d <input file> -q <query file> -k <int> -M <int> -p <int> -o <output file>
```

### 1.2.4 Παράμετροι Hypercube

- $k$  είναι η διάσταση του υπερκύβου
- $M$  είναι το μέγιστο πλήθος σημείων που θα ελέγχονται σε κάθε αναζήτηση
- $p$  είναι το μέγιστο πλήθος κορυφών που θα ελέγχονται ανά αναζήτηση

### 1.2.5 Cluster

```
./cluster -i <input file> -c <configuration file> -o <output file>
```

## 1.3 Input File

Τα αρχεία εισόδου όλων των προγραμμάτων του project πρέπει να έχουν την ακόλουθη μορφή:

x0	0	16	35	5	32	31	14	10	11	78	55	10	45	83	11	6	14	57	...
x1	14	35	19	20	3	1	13	11	16	119	85	5	0	5	24	26	0	27	...
x2	0	1	5	3	44	40	20	14	10	100	63	7	44	47	9	6	7	70	...
x3	12	47	14	25	2	3	4	7	14	122	90	7	0	0	6	14	0	24	...
...																			

```
x0,0,16,35,5,32,31,14,10,11,78,55,10,45,83,11,6,14,57,...
x1,14,35,19,20,3,1,13,11,16,119,85,5,0,5,24,26,0,27,...
x2,0,1,5,3,44,40,20,14,10,100,63,7,44,47,9,6,7,70,...
x3,12,47,14,25,2,3,4,7,14,122,90,7,0,0,6,14,0,24,...
...
```

Κάθε γραμμή δίνει και ένα σημείο του συνόλου δεδομένων. Για κάθε σημείο πρώτα δίνεται το όνομά του και μετά τα στοιχεία του, τα οποία διαχωρίζονται μεταξύ τους με tabs ή κόμματα.

## 1.4 Configuration File

Το configuration file χρησιμοποιείται για την ρύθμιση των παραμέτρων των προγραμμάτων.

### 1.4.1 Πιθανές Παράμετροι

Οι παράμετροι που μπορούν να δοθούν μέσω του configuration file είναι οι εξής:

Παράμετρος	Default	Περιγραφή
number_of_clusters: <int>	-	Πλήθος των clusters που θα δημιουργηθούν
number_of_hash_functions: <int>	4	Πλήθος hash functions του LSH
number_of_hash_tables: <int>	5	Πλήθος hash tables του LSH
hypercube_dimension: <int>	5	Διάσταση του Hypercube
hypercube_max_point_checks: <int>	5000	Μέγιστο πλήθος σημείων που ελέγχει ο Hypercube
hypercube_max_vertex_checks: <int>	20	Μέγιστο πλήθος κορυφών που ελέγχει ο Hypercube
max_iterations: <int>	100	Μέγιστο πλήθος επαναλήψεων στο clustering
metric: <string>	euclidean	Μετρική απόστασης
initialise: <string>	random	Αλγόριθμος αρχικοποίησης στο clustering
assign: <string>	lloyds	Αλγόριθμος ανάθεσης στο clustering
update: <string>	kmeans	Αλγόριθμος ανανέωσης συστάδων στο clustering

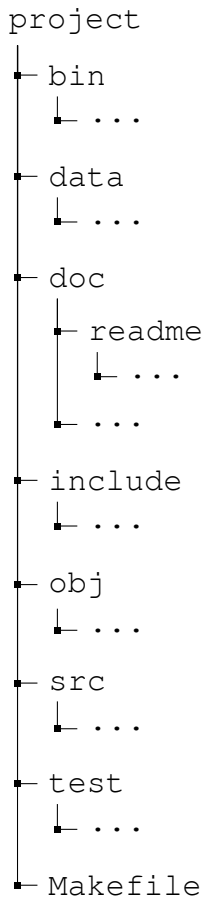
## 2 Εργαλεία Ανάπτυξης Project

### 2.1 Version Control (Git/Github)

Για την καλύτερη διαχείριση των εκδόσεων του κώδικα και των αλλαγών χρησιμοποιείται το πρόγραμμα git και η πλατφόρμα Github.

### 2.2 Unit Testing (CppUnit)

### 3 Οργάνωση Αρχείων & Φακέλων



Ο κώδικας οργανώνεται σε διαφορετικά αρχεία ανάλογα με το σκοπό και τη λειτουργικότητά του. Συγκεκριμένα στα εξής αρχεία:

#### 3.1 Οργάνωση Φακέλων

##### 3.1.1 Φάκελος bin

Περιέχει τα τελικά εκτελέσιμα αρχεία που παράγονται με τη μεταγλώττιση του κώδικα.

##### 3.1.2 Φάκελος data

Περιέχει όλα τα σύνολα δεδομένων που χρησιμοποιούνται.

##### 3.1.3 Φάκελος doc

Περιέχει τα αρχεία που περιγράφουν το project (readme, configuration files).

##### 3.1.4 Φάκελος include

Περιέχει όλα τα αρχεία κεφαλίδας (.hpp) των αρχείων κώδικα που αναπτύχθηκαν για το project.

##### 3.1.5 Φάκελος obj

Περιέχει τα object files που παράγονται κατά τη μεταγλώττιση του κώδικα.

##### 3.1.6 Φάκελος src

Περιέχει όλα τα αρχεία πηγαίου κώδικα (.cpp) που αναπτύχθηκαν για το project.

##### 3.1.7 Φάκελος test

#### 3.2 Οργάνωση Αρχείων Κώδικα

##### 3.2.1 Αρχείο lsh.cpp

Περιέχει την συνάρτηση main που εκτελεί το LSH πάνω σε δοσμένο dataset και εκτυπώνει αποτελέσματα και στατιστικά.

##### 3.2.2 Αρχείο cube.cpp

Περιέχει την συνάρτηση main που εκτελεί τον υπερκύβο πάνω σε δοσμένο dataset και εκτυπώνει αποτελέσματα και στατιστικά.

##### 3.2.3 Αρχείο cluster.cpp

Περιέχει την συνάρτηση main που εκτελεί το cluster creator πάνω σε δοσμένο dataset και εκτυπώνει αποτελέσματα και στατιστικά.

### 3.2.4 Αρχείο searcher.cpp/.hpp

Περιέχουν τη virtual κλάση searcher και την υλοποίηση των υποκλάσεων της, lsh και hypercube.

### 3.2.5 Αρχείο clusterCreator.cpp/.hpp

Περιέχουν την υλοποίηση των αλγορίθμων clustering και του αλγορίθμου αξιολόγησης silhouette.

### 3.2.6 Αρχείο hash\_table.cpp/.hpp

Περιέχει την υλοποίηση των πινάκων κατακερματισμού που χρησιμοποιεί ο αλγόριθμος LSH

### 3.2.7 Αρχείο hasher.cpp/.hpp

Περιέχει την υλοποίηση των μετρικών που χρησιμοποιούνται και από τις δύο μεθόδους (ευκλείδεια απόσταση και συνημιτόνου)

### 3.2.8 Αρχείο point.cpp/.hpp

Περιέχει την υλοποίηση του ATΔ point το οποίο αναπαριστά τα σημεία των datasets και τα διανύσματα των αλγορίθμων

### 3.2.9 Αρχείο util.cpp/.hpp

Περιέχει γενικές συναρτήσεις που χρησιμοποιούνται και από τις δύο μεθόδους

## 4 Approximate Nearest Neighbor

### 4.1 Latent Semantic Hashing (LSH)

### 4.2 Υπερκύβος

## 5 Σύγκριση Μεθόδων LSH & Hypercube

Για την σύγκριση των δύο μεθόδων θα χρησιμοποιήσουμε την **μετρική συνημιτόνου**.

### 5.1 Ενδεικτικές Εκτελέσεις

Στη συνέχεια βλέπουμε τις επιδόσεις των μεθόδων για διαφορετικές παραμέτρους. Είναι πολύ σημαντικό να σημειωθεί ως σημείο αναφοράς ότι ο ντετερμινιστικός αλγόριθμος πλησιέστερου γείτονα τρέχει κατά μέσο όρο σε χρόνο 0.05s. Χρησιμοποιούμε ένα dataset 10000 σημείων ως είσοδο και ένα dataset 100 σημείων για queries.

### 5.1.1 LSH

$k$	$L$	Max Approx.	Avg. Approx	Avg. Time
3	5	1	1	0.2292
4	5	1.0559	1.0011	0.2089
6	5	1.1177	1.0050	0.1469
8	5	1.1070	1.0058	0.0900
10	5	1.5502	1.0367	0.0456
12	5	1.8626	1.0400	0.0416
14	5	2.1293	1.0489	0.0298
15	5	2.1771	1.0786	0.0202
16	5	2.3813	1.0956	0.0274
16	6	2.1134	1.0783	0.0421
16	7	2.0756	1.0808	0.0357
16	8	1.8307	1.0690	0.0235
<b>16</b>	<b>9</b>	<b>1.7638</b>	<b>1.0438</b>	<b>0.0341</b>
16	10	1.6221	1.0505	0.0464

### 5.1.2 Hypercube

$k$	$M$	$p$	Max Approx.	Avg. Approx	Avg. Time
4	500	4	5.0334	1.3621	0.0027
5	500	4	4.9590	1.4644	0.0025
6	500	4	3.3547	1.2262	0.0024
6	1000	4	3.1239	1.1584	0.0050
6	1000	6	1.7797	1.0844	0.0054
6	1000	10	2.4349	1.0989	0.0052
6	2000	10	2.6356	1.0793	0.0069
7	3000	15	2.2929	1.0542	0.0119
7	3000	32	1.7262	1.0447	0.0131
<b>7</b>	<b>3000</b>	<b>64</b>	<b>1.6721</b>	<b>1.0509</b>	<b>0.0138</b>
5	3000	15	2.2484	1.0857	0.0149
5	3000	32	1.8530	1.0498	0.0150

### 5.1.3 Παρατηρήσεις & Σχόλια

Από τις παραπάνω εκτελέσεις παρατηρούμε ότι και οι δύο μέθοδοι καταφέρνουν αρκετά καλά αποτελέσματα με σχετικά παρόμοια ακρίβεια (συγκεκριμένα για  $k = 16, L = 9$  στο LSH και  $k = 7, M = 3000, p = 64$  στον υπερκύβο). Παρατηρούμε όμως ότι η μέθοδος του υπερκύβου είναι περίπου 2 φορές πιο γρήγορη από το LSH και περίπου 3 φορές πιο γρήγορη από τον ντετερμινιστικό αλγόριθμο.

Επίσης, για το LSH παρατηρούμε ότι δεν έχει νόημα η παράμετρος  $k$  να παίρνει τιμές κάτω από 14 για τη συγκεκριμένη μετρική, διότι ο χρόνος που πετυχαίνει είναι χειρότερος από τον ντετερμινιστικό αλγόριθμο. Αυτό συμβαίνει διότι η συνάρτηση κατακερματισμού έχει πολλές συγκρούσεις για μικρά  $k$ .

Ακόμη, στον υπερκύβο παρατηρούμε ότι είναι σημαντικό όταν αυξάνεται το  $k$  να αυξάνεται και το  $p$ , το οποίο είναι λογικό αφού όταν έχουμε πολλές κορυφές θα πρέπει να επιτρέπουμε στον αλγόριθμο να επισκέπτεται περισσότερες.

## 5.2 Κατανάλωση Χώρου

### 5.2.1 LSH

Για την υλοποίηση του LSH χρησιμοποιούνται  $L$  hash tables, καθένα απο τα οποία χρησιμοποιεί τη δική του συνάρτηση κατακερματισμού και κάθε μια απ' αυτές έχει  $k$  "υπο-συναρτήσεις" με ξεχωριστά διανύσματα ανάλογα τη μετρική.

Για κάθε hash table δημιουργούνται  $k$  buckets, τα οποία αναπαρίστανται με C++ vectors και σε αυτά αποθηκεύονται οι δείκτες των σημείων που είναι αποθηκευμένα σε μια εξωτερική δομή για την αποφυγή της πολλαπλής (για την ακρίβεια  $k$ -πλής) αποθήκευσης των σημείων.

### 5.2.2 Υπερκύβος

Στην περίπτωση του υπερκύβου, έχουμε  $2^k$  κορυφές οι οποίες έχουν ίδια μορφή με τα buckets των hash tables (C++ vectors με pointers σε σημεία), μόνο μια συνάρτηση κατακερματισμού με  $2^k$  υποσυναρτήσεις και  $k$  συναρτήσεις  $f : x \rightarrow [f_1(h_1(x)), \dots, f_{2^k}(h_{2^k}(x))] \in \{0, 1\}^{2^k}$ . Για την υλοποίηση των συναρτήσεων  $f$  χρησιμοποιείται η δομή `unordered_map` της C++ όπου κάθε τιμή που δεν περιέχεται ήδη στο `map` προβάλεται τυχαία στο  $\{0, 1\}$ .

## 6 Σύγκριση Μεθόδων Clustering