# National and Kapodistrian University of Athens

Department of Computer Science

Software Development for Algorithmic Problems

K23$\gamma$

## Hashing and Search over vectors and polygonal curves

---

*STUDENT ID :*
sdi1600151
sdi1700115

# Contents

# 1 ABSTRACT

In this project, the above algorithms have been implemented: LSH and Hypercube of vectors, Grid LSH and Hypercube - Random Projection LSH and Hypercube of curves.

# 2 Directories and Files

Our implementation code is derived in the following directories:

- core: includes functions called from all interfaces
  - hash: includes hash function's and amplified hash function's implementation
  - metric: includes the implementation of different metrics (ex:DTW,Manhattan,Euclidean)
  - search: includes the implementation of functions commonly used by different search algorithms in different interfaces.
  - utils: includes the implementation of interfaces' utilisation.
- curves
  - grid_lsh: includes the implementation of grid lsh algorithm.
  - grid_hypercube: includes the implementation of grid hypercube algorithm.
  - projection_lsh: includes the implementation of projection lsh algorithm.
  - projection_hypercube: includes the implementation of projection hypercube algorithm.
- datasets: includes curves' and vectors' dataset and query files.
- results: includes search algorithms' results
- scripts: includes compilation script for each interface
- vectors
  - lsh: includes the implementation of lsh algorithm.
  - hypercube: includes the implementation of hypercube algorithm.

# 3 LSH - Vectors

## Compilation and Running

Navigate to **scripts** file, then run **sh run_lsh.sh**. Modify compilation scripts in case you want to change arguments. Argument Parameters:

- -d input file
- -q query file
- -K number of LSH hash functions for each hash table
- -L number of LSH hash tables
- -o output file

Default hyperparameters: $K = 4, L = 5$.

## Implementation Details

- All arguments are validated. In case of no arguments, user is asked to provide them from input.
- All d-dimensional points are stored in in 1D vector allongside with their offsets and ids, gaining a better performance.
- Radius is calculated as the average distance between vectors' points.
- Window is calculated as $4 \times Radius$.
- Except from Exact NN we have also implemented a Radius NN Search.
- Metric: Manhattan Distance.

## Results

Comparing the results of Exact Nearest Neighbor using Brute Force and LSH we get the above - (with default parameters):

- Max Average Fraction: 2.06527
- Avg Average Fraction: 1.096
- Not found: 0

- Time elapsed - NN execution: 0.0296563 seconds

# 4 Hypercube - Vectors

## Compilation and Running

Navigate to **scripts** file, then run **sh run_hypercube.sh**. Modify compilation scripts in case you want to change arguments. Argument Parameters:

- -d input file
- -q query file
- -M max candidate points to be checked
- –probes max vertices to be checked
- -k dimensional space reduction
- -o output file

Default hyperparameters: $k = 3, M = 10, probes = 2$.

## Implementation Details

- All arguments are validated. In case of no arguments, user is asked to provide them from input.
- All d-dimensional points are stored in in 1D vector allongside with their offsets and ids, gaining a better performance.
- Radius is calculated as the average distance between vectors' points.
- Window is calculated as $2 \times Radius$.
- Except from Exact NN we have also implemented a Radius NN Search.
- Metric: Manhattan Distance.

## Results

Comparing the results of Exact Nearest Neighbor using Brute Force and LSH we get the above - (with default parameters):

- Max Average Fraction: 2.32817
- Avg Average Fraction: 1.11314
- Not found: 0
- Time elapsed - NN execution: 0.0418538 seconds

# 5 Grid LSH - Curves

## Compilation and Running

Navigate to **scripts** file, then run **sh run_grid_lsh.sh**. Modify compilation scripts in case you want to change arguments. Argument Parameters:

- -d input file
- -q query file
- –k_vec number of LSH hash functions for each hashTable
- –L_grid number of LSH Structures
- -o output file

Default hyperparameters: $K\_vec = 4, L\_grid = 5$.

## Implementation Details

- All arguments are validated. In case of no arguments, user is asked to provide them from input.
- All d-dimensional pairs are stored in in 1D vector allongside with their offsets,ids and lengths - as curves do not have the same length - gaining a better performance.
- For each curve into dataset we calculate the average euclidean distance of consecutive pairs of points. Delta equals the average of all the above values $\times$ a factor (default = 10).
- Radius equals with the average exact distance per Grid. So, each structure has a different radius.
- Window equals r $\times$ factor.

- Metric: Dynamic Time Wrapping.

### Results

Comparing the results of Exact Nearest Neighbor using Brute Force and LSH we get the above - (with default parameters):

- Max Average Fraction: 3.12161
- Avg Average Fraction: 1.07941
- Not found: 1
- Time elapsed - NN execution: 15.2301 seconds

## 6 Grid Hypercube - Curves

### Compilation and Running

Navigate to **scripts** file, then run **sh run_grid_hypercube.sh**. Modify compilation scripts in case you want to change arguments. Argument Parameters:

- -d input file
- -q query file
- –k_vec number of LSH hash functions for each hashTable
- -M max candidate points to be checked
- –probes max vertices to be checked
- –L_grid number of LSH Structures
- -o output file

Default hyperparameters: $K\_vec = 4, L\_vec = 5, M = 10, probes = 2$.

### Implementation Details

- All arguments are validated. In case of no arguments, user is asked to provide them from input.

- All d-dimensional pairs are stored in in 1D vector allongside with their offsets,ids and lengths - as curves do not have the same length - gaining a better performance.
- For each curve into dataset we calculate the average euclidean distance of consecutive pairs of points. Delta equals the average of all the above values $\times$ a factor (default = 10).
- Radius equals with the average exact distance per Grid. So, each structure has a different radius.
- Window equals $r \times factor$.
- Metric: Dynamic Time Wrapping.

### Results

Comparing the results of Exact Nearest Neighbor using Brute Force and LSH we get the above - (with default parameters):

- Max Average Fraction: 2.75153
- Avg Average Fraction: 1.00866
- Not found: 0
- Time elapsed at NN execution: 41.205 seconds

## 7 Random Projection LSH - Curves

### Compilation and Running

Navigate to **scripts** file, then run **sh run_projection_lsh.sh**. Modify compilation scripts in case you want to change arguments. Argument Parameters:

- -d input file
- -q query file
- –k_vec number of LSH hash functions for each hashTable
- –L_vec number of LSH Structures
- -e error rate

- -o output file

Default hyperparameters: $K\_vec = 4, L\_grid = 5, e = 0.5$.

## Implementation Details

- All arguments are validated. In case of no arguments, user is asked to provide them from input.
- All d-dimensional pairs are stored in in 1D vector allongside with their offsets,ids and lengths - as curves do not have the same length - gaining a better performance.
- Radius is calculated as the average distance between curves' points.
- We consider as relevant traversal the combination of pairs in diagonal or upper diagonal.
- Not all relevant traversals are calculated (space reduction).
- All traversals are stored in a $M\times$ matrix, where M is max length of curves.
- Each dataset's curve is mapped into all relevant traversals of cell[length][*].
- Each query curve is mapped into all relevant traversals of cell[*][length].
- Each vector as well as its information is stored in a map with a tuple as key.
- An LSH structure is build per relevant traversal.
- Metric: Dynamic Time Wrapping.

## Results

Comparing the results of Exact Nearest Neighbor using Brute Force and LSH we get the above - (with default parameters):

- Max Average Fraction: 60.8
- Avg Average Fraction: 4.52
- Time elapsed at NN execution: 0.120878 seconds

# 8 Random Projection Hypercube - Curves

## Compilation and Running

Navigate to **scripts** file, then run **sh run_projection_hypercube.sh**. Modify compilation scripts in case you want to change arguments. Argument Parameters:

- -d input file
- -q query file
- –k_vec number of LSH hash functions for each hashTable
- –L_vec number of LSH Structures
- -M max candidate points to be checked
- –probes max vertices to be checked
- -e error rate
- -o output file

Default hyperparameters: $K\_vec = 4, L\_grid = 5, M = 10, probes = 2$.

## Implementation Details

- All arguments are validated. In case of no arguments, user is asked to provide them from input.
- All d-dimensional pairs are stored in in 1D vector allongside with their offsets,ids and lengths - as curves do not have the same length - gaining a better performance.
- Radius is calculated as the average distance between curves' points.
- We consider as relevant traversal the combination of pairs in diagonal or upper diagonal.
- Not all relevant traversals are calculated (space reduction).
- All traversals are stored in a M $\times$ matrix, where M is max length of curves.
- Each dataset's curve is mapped into all relevant traversals of cell[length][*].
- Each query curve is mapped into all relevant traversals of cell[*][length].

- Each vector as well as its information is stored in a map with a tuple as key.
- A Hypercube structure is build per relevant traversal.
- Metric: Dynamic Time Wrapping.

### Results

Comparing the results of Exact Nearest Neighbor using Brute Force and LSH we get the above - (with default parameters):

- Max Average Fraction: 62.4
- Avg Average Fraction: 7.23
- Time elapsed at NN execution: 0.209213 seconds

## 9 Conclusion

Regarding the above results we conclude that LSH finds more accurate nneighbors than Hypercube. Nevertheless, these algorithms are extremely sensitive, meaning that results depend not only to parameters chosen but also to input dataset. Furthermore, our results in random projection do not seem good enough due to the reduction of relevant traversals. In general, random projection approach might need more space complexity but regarding time complexity - it is asymptotically more efficient than Grid approach. On the other hand, Grid approach achieves good results but it lacks performance.