National and Kapodistrian University of Athens
Department of Informatics and Telecommunications

K23α: Software Development for Information Systems

Exercise #2

Winter Semester 2024-2025

Team Members	**Student Numbers**
Michael-Raphael Kostagiannis	1115202100078
Giorgos Sofronas	1115202100180
Styliani Tziouma	1115202100184

Introduction

All the project requirements have been fulfilled. All programs have been thoroughly tested with various arguments. Neither the programs, nor the unit tests have any memory leaks.

Sample executions (found in the provided scripts) produce astonishing results, with an average **recall of 99.70%** and total execution time of $\sim$ **27 seconds**, for the **filtered vamana** executable. Respectively, the **stitched vamana** executable produces an average **recall of 99.96%** and total execution time of $\sim$ **31 seconds**.

The project's workload was equally distributed amongst our team members. Specifically, here is a comprehensive list of each member's work:

- Michael:
  - Directed Graph: stitch
  - Filtered Robust Prune
  - Creation of groundtruth file
- Giorgos:
  - Filtered Vamana
  - Find Medoid
  - Scripts
- Styliani:
  - Filtered Greedy Search
  - main
  - Stitched Vamana

Compilation and Execution

Along with the source and header files, found under `src/` and `include/` respectively, the project is instrumented using a group of three Makefiles:

1. Present under the `src/` directory, responsible for the compilation of the two main executables (`filtered` and `stitched`) and the `groundtruth` executable.
2. Can be found under the `tests/` directory. It is responsible for the compilation of the unit tests.
3. At the root of the project, from which both aforementioned makefiles are being controlled.

As far as users are concerned, **they should only use the makefile at the root of the project** to compile the executable programs and/or the unit tests.
The available commands are listed in the following table:

Compilation of FilteredVamana:	`make filtered`
Compilation of StitchedVamana:	`make stitched`
Compilation of groundtruth:	`make groundtruth`
Compilation of the unit tests:	`make tests`
Compilation of all of the above:	`make`
Cleanup command:	`make clean`

Object files are stored inside a real-time created directory called `build/`.

Both the FilteredVamana and StitchedVamana are executed using the following common mandatory flags:

- -b: The base vectors file.
- -q: The query vectors file.
- -g: The ground truth vectors file.
- -n: The number of vectors found in the base vectors file.
- -m: The number of valid vectors found in the query vectors file. Valid query vectors have a query type of 0 or 1.
- -a: The distance threshold used by Filtered Vamana. Must be greater than 1.
- -L: Search list size used by Filtered Greedy Search. Must be greater or equal to k.
- -t: Tau threshold used by Find Medoid. Cannot be less than 1.
- -i: The index of a query, used to calculate recall. User can input -1 to calculate total recall.

The FilteredVamana program also needs the following mandatory flag:

- -R: The degree bound used by Filtered Robust Prune.

Here is a sample execution:

```
./filtered -b dummy/dummy-data.bin -q dummy/dummy-queries.bin -g dummy/dummy-groundtruth.bin
-n 10000 -m 5012 -a 1.1 -L 150 -R 12 -t 50 -i -1
```

The StitchedVamana program also needs the following mandatory flags:

- -l: L_{small} used by Stitched Vamana.
- -r: R_{small} used by Stitched Vamana.
- -R: $R_{stitched}$ used by Stitched Vamana.

Here is a sample execution:
```
./stitched -b dummy/dummy-data.bin -q dummy/dummy-queries.bin -g dummy/dummy-groundtruth.bin
-n 10000 -m 5012 -a 1.1 -L 150 -l 100 -r 32 -R 64 -t 50 -i -1
```

Both of the above executables can also use the following optional flags:

- -s: The name of the binary file to save the Vamana graph to.
- -v: The name of the binary file to load the Vamana graph from.

Here are some sample executions:
```
./filtered -b dummy/dummy-data.bin -q dummy/dummy-queries.bin -g dummy/dummy-groundtruth.bin
-s vamana.bin -n 10000 -m 5012 -a 1.1 -L 150 -R 12 -t 50 -i -1

./filtered -b dummy/dummy-data.bin -q dummy/dummy-queries.bin -g dummy/dummy-groundtruth.bin
-v vamana.bin -n 10000 -m 5012 -a 1.1 -L 150 -R 12 -t 50 -i -1
```

All flags (both mandatory and optional) can be given in any order.

Unit Testing

All of our functions and methods have been extensively tested, using acutest: a simple header-only C/C++ unit testing facility. The tests are focused on checking a wide range of both common and edge cases to ensure the program's reliability and results. All of our tests can be found under the `tests/` directory.

To simplify the tests' execution, a simple bash script (`scripts/run-tests.sh`) is provided, with which all the unit tests are executed at once. This script is also used as an automated testing tool in our GitHub repository, whenever a push or a pull-request occurs (GitHub Actions).

Scripts

A wide range of bash scripts is provided under the `scripts/` directory. More specifically, there are two types of scripts:

1. `run-`X`.sh`: Executes X. For both vamana executables, this includes three different executions with different parameter options. For the unit tests, all tests are executed back-to-back.
2. `valgrind-`X`.sh`: Same instructions as the corresponding `run-`X`.sh` script, but with valgrind to ensure that there are no memory leaks.

***Note*: The scripts should be executed from the main/base directory of the project for the relative-paths inside them to be valid.**
Example: execution of filtered-vamana run: `./scripts/run-filtered.sh`

Vectors class

The `Vectors` template class handles vector collections and provides quick and easy access to their distances. Same implementation with the previous report.

Directed Graph class

The implementation of the Directed Graph class remains the same. The only new addition is the `stitch(DirectedGraph *g, int *Pf)` method. This method stitches a graph `g` created with the P_f dataset to an existing graph. This is simply done by unionizing their edge sets.

utils.hpp

Contains a useful utility function commonly found in most files: `ERROR_EXIT(cond, msg)` is a preprocessor macro which terminates the program and prints the given error message whenever the condition is true.

Filtered Greedy Search(G, base_vectors, s, x_q, k, L)

The code of filtered greedy search closely follows the algorithm with the following modifications for performance optimization:

- **L_set**: Is represented by an ordered set according to Euclidean distance from the query, for easier access to the closest vectors.
- **visited**: Is represented by a boolean array for faster determination of index visitability.
- **return values**: As the visited set, the $L_$set is returned, which after the while-loop contains only visited nodes. Due to the constraint on $L_$set being limited to a length of L, some visited nodes may have been deleted. Thus, before returning, using the visited array, they are reinserted into $L_$set.
- **S**: The initial nodes are not represented as an array but as a single node, as filtered greedy search is always initiated with a single starting node.

Filtered Robust Prune(G, p, V, a, R)

Modifies graph 'G' by setting at most 'R' new out-neighbors for point with index 'p' and 'a' distance threshold. The only difference is that 'V' is now an ordered candidate set containing pairs of (euclidean distance, index) and it is sorted in ascending euclidean distance between each point and point 'p'. We do this to calculate p^* in $O(1)$ time (it is the first element of 'V') instead of manually calculating it in $O(n)$ time. Moreover, all euclidean distances are cached inside our distance matrix, so the comparison $\alpha \cdot d(p^*, p') \leq d(p, p')$ is also done in $O(1)$ time. This version of robust prune also takes filters into consideration.

Find Medoid

The `find_medoid` function is based on the pseudocode of the provided paper. In this assignment it is known that each point is matched with at most one filter. Leveraging this fact, `find_medoid`'s implementation is slightly altered from the one in the paper. Specifically, the `T` counter-map isn't implemented since it isn't useful based on the aforementioned knowledge.

Filtered Vamana

This function constructs a directed graph using the Filtered Vamana algorithm. The algorithm operates on a set of vectors and progressively builds the graph by iteratively connecting points based on filter-based constraints.

The code of filtered vamana follows the provided pseudocode of the paper.

Stitched Vamana

The code of stitched vamana closely follows the algorithm with the following modification for performance optimization. In the first for loop, if the filter corresponds to only one base vector, Vamana is not called because there are no other vectors to construct the graph.

GreedySearch, RobustPrune and Vamana

Since StitchedVamana calls Vamana for a specific subset $P_f \subseteq P$, the above functions have been updated to also support P_f.

main.cpp

The main function handles the arguments appropriately, by executing the relevant functions and printing the recall.

As it may have already been observed, both the `filtered` and `stitched` vamana executables are generated from a single `main.cpp` file. This functionality was easily implemented with a simple use of `#ifdef, #else, #endif` pre-processor instructions. Since most of the executables' code and actions are similar, this is an elegant and effective way to generate both executables with one `main.cpp` source file.

Groundtruth file creation

The `groundtruth_brute_force.cpp` source file contains the algorithm for calculating the true 100 nearest neighbors of all query vectors using brute force and writes them to a new binary file. It is executed using the following commad:
`./groundtruth <data vectors file> <query vectors file> <groundtruth filename>`
The output file contains 100 integers which are the indices of the 100 true nearest neighbors of the first query vector, followed by 100 integers for the second query vector, and so on. If a query has less than 100 true nearest neighbors, the empty spaces are padded with -1's to make reading easier.