Final Project Report

Team#11:
Chou, Kelvin, 862466579
Yu, Kunyi, 862548836
Ibrahim, Mona, 862267986

Abstract: This report focuses on the code and experiment replication of the provided paper [1] and corresponds to project option 3. It begins with a paper review, covering the summary of query problems, indexing architecture, and the POWER query processing algorithm. Next, the code replication process is introduced, including code decomposition and experiment design. Finally, the experiment results are presented and discussed, followed by the conclusion.

Index Terms: kNN spatial-keyword query, Indexing architecture, Query processing algorithm, Code replication

1. Paper Review

The paper of cutting-edge Spatial-keyword querying process is a 2022 published paper by Yongyi Liu and Amr Magdy [1]. Main contributions of the paper includes:

1) Propose two novel kNN spatial-keyword query problems
   - TKQN: Top-k kNN Query with Negative keyword predicates (we choose this)
   - BKQN: Boolean kNN Query with negative keyword predicates
2) Introduce U-ASK: a unified architecture for spatial-keyword query with negative keyword predictions
3) Introduce POWER: a query processing algorithm handling TKQN and BKQN queries
4) Present experimental evaluation on real datasets

Current work of the spatial-keyword querying process has two limitations: one is lack of dealing with phrases with a sequence of words, and the other one supports only boolean kNN query. To handle the limitations, U-ASK proposed by the paper supports two types of kNN spatial-keyword queries with AND, OR, and NOT conjunctions, which consists of an index framework TEQ (Textual Enhanced Quadtree) and a query processing algorithm POWER (Parallel Bottom-up Search with Incremental Pruning).

The problem of TKQN takes a tuple of location, positive/negative keywords, weight factor, and number k, then outputs top-k output with at least one positive keyword, without any negative keyword, and ranked by a spatial-textual-score function. The input format is as follows:

$$q_t = (q_t.loc, q_t.pos, q_t.neg, q_t.\lambda, q_t.k)$$

The score function is defined as:

$$score(o_i, q_t) = q_t.\lambda * score_s(o_i, q_t) + (1 - q_t.\lambda) * score_t(o_i, q_t)$$

Moreover, TEQ indexing is a hybrid, memory-resident index for TKQN queries which combines the strengths of a quadtree for spatial partitioning and an inverted index for keyword organization. Structure: each leaf cell n in the quadtree contains 4 components:

1) *n.ltp*: a location table pointer to a hash file on disk
2) *n.neigh*: a list of spatial neighboring cells
3) *n.iti*: an inverted textual index (hashtable) mapping keyword w to the tuple:
    a) *w.size*: number of objects containing the keyword w
    b) *w.max*: maximum weight of w in the cell
    c) *w.listPtr*: pointer to the sorted inverted list file
    d) *w.setPtr*: pointer to the sorted inverted set file (faster boolean filtering)
4) *n.oti*: an object textual index (hashtable) mapping object IDs to full text

The way of constructing TEQ contains two passes. The first pass builds the spatial indexing component: (build tree)

1) Insert objects into the quadtree
2) Quadtree cells are split based on object density
3) Create *n.neigh* and *n.ltp*

The second pass builds the textual indexing component: (build cells)

1) Initialize two hashtables: *n.iti* and *n.oti*
2) Insert every object o into *n.oti*
3) Insert every keyword w with its weight into *n.iti*
4) Sort inverted list and inverted set, store all parameters into disk

The query problem our group chose is TKQN, so as requirements, we need to implement the POWER algorithm. The POWER (Parallel bOttom-up search With incrEmental pRuning) operates within a master-worker framework, where each worker processes local top-k searches within assigned index cells. First, the POWER will process index cell by loading location table *n.LT* into a buffer with LRU policy (prepared for parallelization, the project will omit it). Second, the POWER uses a top-k search strategy based on the Threshold Algorithm (TA), which incrementally retrieves and aggregates scores from multiple sorted lists. Thanks to the upper bound score, it will prune many useless searches. Third, because the TKQN query ranks its results based on textual attribute (can be found in keyword-inverted lists) and spatial attribute (sorted based on the query location), a priority queue will incrementally retrieve top-k objects based on the TA algorithm mentioned before. Lastly, the POWER will evaluate both positive keyword predictions (*q.pos*) and negative keyword predictions (*q.neg*) by a series rigorous steps. Note that, due to the computationally expensive nature of using a priority queue, the paper also introduces POWER-T (textual pruning) and POWER-S (spatial-pruning) to reduce query cost, but the project option 3 will not cover this part.

The experimental evaluation is in section 6 of the paper, which contains 6.1) experimental setup, 6.2) performance evaluation for the different parameters and framework components, and 6.3/4) compares the proposed algorithms against the SOTA under TKQN and BKQN, respectively. In addition to 6.1) experimental setup, the experiments suitable for this project are mainly distributed in 6.3) TKQN query evaluation.

For the 6.1) experimental setup, there is a summary in the "Table 2: Evaluation Parameters Values" of the paper.

For the 6.3) TKQN query evaluation, 4 experiments are conducted:

1) The effect of query keywords: change different number of positive words ($|q.pos|$), number of negative phrases ($|q.neg|$), and length of negative phrases ($q.negLen$), showing how the performance floats;

2) The effect of weighting factor: change different weighting factors (λ), showing how the balance between spatial and textual scores affects performance;
3) The effect of k: change different answer sizes ($q.k$), showing how the multi-threading master-worker architecture of POWER-T affects;
4) The effect of dataset size: change different dataset sizes, showing how the scalability varies.

The figures of above experiments' results can be found in the "Figure 5: TKQN Query Evaluation" of the paper.

2. Code Replication

In our code replication process, we choose Python as programming language and manage with multi files to decompose the complexity of the project. The python source files are stored in directory *./src* and the structure is as follows:

```
./src
 exps
     exp_a.py
     exp_b.py
     exp_c.py
     exp_d.py  // deprecated, no lambda for the POWER algorithm
     exp_e.py
     exp_f.py
 invertedIndex.py
 main.py
 plots.py
 point.py
 power.py
 power_batch.py
 quadTreeNode.py
 query.py
 read_data.py
```

The main entry of the project is *main.py*, which contains the main function to run the experiments. The *read_data.py*, *point.py*, *quadTreeNode.py*, *invertedIndex.py*, *query.py*, *power.py*, and *power_batch.py* are the core modules of the project. The implementation logic are follow the paper closely. Note that the $power_batch.py$ is used to run the experiments in batch mode, which will first gether the queries with the same parameters into a group and then run the POWER algorithm for each group.

The *plots.py* is used to generate the plots for the experiments. The *exps* directory contains the experiments mentioned in the paper subsection 6.3. The python files in the *exps* directory will be called by the *main.py* to run the experiments.

Instruction to run the code:

```
$ python main.py --exp=<exp_name> --size=<dataset_size>
$ # exp_name: a, b, c, d, e, f (default a)
$ # dataset_size: 2, 4, 6, 8, 10 (default 4)
```

3. Experiment Results

4. Conclusion

Acknowledgements

The authors wish to thank Professor Amr Magdy, Dr. Yongyi Liu, TA Alhassan Satii Alshareedah, and others who have helped us in this course for their valuable suggestions and patient guidance. Wish them all the best in their future work and life!

The contribution of each team member is as follows:

- Chou, Kelvin
  - Paper reading
- Yu, Kunyi
  - Paper reading
  - Assignment 3/5 report, final report
  - Code decomposition, experiments implementation
- Ibrahim, Mona
  - Paper reading
  - First/second version code

References

[1] Liu, Yongyi, and Amr Magdy. "U-ASK: a unified architecture for kNN spatial-keyword queries supporting negative keyword predicates." Proceedings of the 30th International Conference on Advances in Geographic Information Systems. 2022.