Implementation of Efficient Similarity Search Algorithm for Spatiotemporal Scenarios

GRADUATE PROJECT REPORT

Submitted to the Faculty of
the Department of Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

By

Akhil Reddy Kothapally
Spring 2018

Committee Members

Dr. Junfei Xie
Committee Chairperson

Dr. Ajay Katangur
Committee Member

ABSTRACT

The advance of sensing technology allows us to collect vast amount of spatiotemporal data that include both spatial and temporal information. To extract useful information from the data saved in the database so as to facilitate decision making, efficient query techniques are required. In this work, an efficient similarity search algorithm for spatiotemporal scenario data, which is a new data type, is reviewed and implemented. This similarity search algorithm is prototyped in Matlab and implemented in Java for higher efficiency. Comparative simulation studies show the efficiency of the Java-based implementation.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The spatiotemporal scenario data, representing a new data type, is typically generated from physical processes of spatiotemporal evolving dynamics. Such data are featured by spatiotemporally correlated dynamics of spread patterns with changing shape, size, location, and intensity. The K-nearest neighbor similarity search problem is concerned with *finding the top-K most similar spatiotemporal scenarios to a query scenario"*, where K is a user provided input. To similarity search, a distance measure that quantifies the similarity/difference between two spatiotemporal scenarios is needed.

Many distance measures have been developed for spatiotemporal point data [6, 14, 16, 18, 21, 26]. The most commonly used distance measures over spatiotemporal moving objects include Longest Common Subsequence(LCSS) [26] and Dynamic Time Warping (DTW) [1]. Extended from these distance measures, Fast search method for Dynamic Time Warping (FTW) was proposed in [18] to reduce the computation cost of DTW. The One Way Distance (OWD) Measure was introduced in [14], which demonstrates higher accuracy than DTW. The Edit Distance on Real Sequence (EDR) measure introduced in [6] increases the robustness of the LCSS. These distance measures and many other algorithms such as the Euclidean distance or its variants [2, 11, 20] have also been developed for clustering purpose. However, none of these distance measures can be directly applied to the spatiotemporal scenario data because of the unique spread patterns in the data. To capture the similarity between spatiotemporal scenarios of spread patterns, a 3-D multiresolution spatiotemporal distance measure was developed [24, 25].

With the distance measure, we are able to perform similarity search by comparing the query scenario with each scenario in the database. However, this approach is too computational expansive especially when the database size is larger and/or the distance measure is complicated to calculate. To address this issue, many query processing techniques have been developed to improve the efficiency [3, 4, 5, 8, 9, 12, 13, 15, 17, 19, 22, 26, 27]. B-tree, R-tree, KD tree are the most widely used similarity search structures. These similarity search structures use triangle inequality and bounding surfaces to prune and select objects. Though they give accurate results, the number of objects to search is a variable. If the database is large or if the computation time for distance measure is high, then these structures will take a lot of time to provide exact results. By sacrificing the accuracy and giving priority to efficiency, many approximate similarity search structures have been developed. BD-tree, Locality Sensitive Hashing, and spatial approximate sample hierarchy are some of the examples for approximate similarity search structures. Though these indexing techniques provide good results with high efficiency, they cannot be directly applied over the multiresolution spatiotemporal distance measure to perform similarity search for the spatiotemporal scenario data. This is because of the non-linearity and complexity of the distance measure that compares two scenarios at multiple resolutions.

In [23], a multiresolution spatiotemporal distance based similarity search algorithm was developed, which explores the unique features of the multiresolution spatiotemporal distance measure to quickly find similar scenarios. However, this algorithm was prototyped in Matlab in [23] and reads data directly from the local files. To make this similarity search algorithm applicable for large-scale real-world problems, this project aims to implement this algorithm using Java, store data in

relational databases, and create user-interfaces to allow friendly user-database interaction. To evaluate the performance of the Java-based implementation, comparative simulation studies using real weather forecast data are conducted. The results show the efficiency of the Java-based implementation.

CHAPTER 2

PRELIMINARIES

In this chapter, the multiresolution spatiotemporal distance measure [24, 25] is first reviewed, followed by the description of the similarity search algorithm developed based on this distance measure.

2.1 Multiresolution Spatiotemporal Distance measure

First, let's describe the mathematical formula for the multiresolution spatiotemporal distance measure. Let s_i and s_j be two spatiotemporal scenarios with the same number of spatial cells $g_k \in G$ and time points $t_m \in T$. Here G and T represent a whole set of spatial cells and time points. The equation for overall pairwise scenario distance $D_{i,j}$ between s_i and s_j is given as:

$$D_{i,j} = \sum_{h=1}^{h_{max}} \sum_{w=1}^{w_{max}} d_{i,j,w,h} \frac{\delta_w \alpha_h}{\sum_{h=1}^{h_{max}} \sum_{w=1}^{w_{max}} \delta_w \alpha_h} \tag{2.1}$$

where size of moving windows along spatial and temporal windows is defined by w and h respectively. The maximum spatial and temporal window sizes are defined as w_{max} and h_{max} respectively. δ_w and α_h are spatial and temporal window size weighting factors. Their values decrease as window sizes increase. The values for δ_w and α_h are set to $\delta_w = e^{-\sigma(w-1)}$ and $\alpha_h = e^{-\rho(h-1)}$ where $\sigma, \rho \geq 0$. The distance for a particular spatial resolution w and temporal resolution h is represented as $d_{i,j,w,h}$,

and it can be computed by the following equation:

$$d_{i,j,w,h} = \sum_{\phi_{k,w} \in \Phi_w} \sum_{\phi_{l,h} \in \Phi_h} \frac{1}{|\phi_{k,w}||\phi_{l,h}||\Phi_h|} \left| \sum_{g_n \in \phi_{k,w}} \sum_{t_m \in \phi_{l,h}} \frac{\hat{I}_{i,n,m}}{\lambda_{n,w}\tau_{m,h}} - \sum_{g_n \in \phi_{k,w}} \sum_{t_m \in \phi_{l,h}} \frac{\hat{I}_{j,n,m}}{\lambda_{n,w}\tau_{m,h}} \right| \tag{2.2}$$

Here $\hat{I}_{i,n,m} = \beta_{n,m} I_{i,n,m}$ is the weighted intensity of scenario s_i at cell $g_n \in G$ and at time point $t_m \in T$ and $\beta_{n,m} > 0$ is a constant weight for a spatiotemporal cell and $I_{i,n,m}$ is the actual intensity value of that cell. Φ_w represents the set of all spatial windows and $\phi_{k,w}$ contains a set of all spatial cells that are reachable in $(w-1)$ hops from g_k. Φ_h represents the set of all temporal windows and $\phi_{l,h}$ contains a set of all temporal cells that are reachable in $(h-1)$ hops from t_l. $|A|$ represents the total number of elements present in set A, and A can be $\phi_{k,w}$ or $\phi_{l,h}$ or Φ_h. $\lambda_{n,w}$ represents the spatial contribution for spatial cell g_n and $\tau_{m,h}$ represents the temporal contribution for time point t_m.

2.2 Similarity Search for Spatiotemporal Scenarios

From a given database S and a query scenarios S_q, the similarity search problem is concerned with finding the top `K` scenarios from the database that are most similar to S_q. This is achieved using the multiresolution spatiotemopral distance measure algorithm mentioned in 2.1. In particular, using this distance measure as filtering criteria, the multiresolution spatiotemporal distance based similarity search algorithm prunes the search space and selects top-`K` scenarios from the database.

In order to prune the search space at each iteration of scenario comparison, we find the upper and lower bounds of the distance measure, and use these bounds to alter out the dissimilar scenarios. Suppose $\underline{D}_{i,j}$ and $\overline{D}_{i,j}$ are lower and upper bounds of $D_{i,j}$. For a given scenario s_q, let $A_{\mathtt{K}} = \{s_i\}$, $A_{\mathtt{K}} \subseteq S$, be the set of `K` scenarios with

the smallest upper bound values, i.e., $\overline{D}_{i,q} \leq \overline{D}_{j,q}$ for all $s_i \in A_{\mathtt{K}}$ and $s_j \in S \setminus A_{\mathtt{K}}$. Let $M_{\mathtt{K}}$ be the largest upper bound value in $A_{\mathtt{K}}$ (corresponding to the K-th smallest upper bound value in S), i.e., $M_{\mathtt{K}} = \max \overline{D}_{i,q}$, $s_i \in A_{\mathtt{K}}$. We can then safely discard all scenarios $s_i \in S$ that satisfy the following condition:

$$\underline{D}_{i,j} > M_k \tag{2.3}$$

2.2.1 Lower and Upper bounds of Multiresolution SpatioTemporal Distance Measure

The pairwise distance $D_{i,j}$ from equation 2.1 is a weighted sum of distance measure $d_{i,j,w,h}$ computed from equation 2.2 at each spatial resolution w and temporal resolution h, where $1 \leq w \leq w_{max}$ and $1 \leq h \leq h_{max}$.

The upper bound of $D_{i,j}$ can be gradually reduced when distances $d_{i,j,w,h}$ are calculated at coarser resolutions. In particular, suppose a spatiotemporal window of size $w = n$ and $h = m$ is used to scan the scenarios at the k-th iteration, where $1 \leq n \leq w_{max}$, $1 \leq m \leq h_{max}$, and $k \in \{1, 2, ..., w_{max}h_{max}\}$. We can gradually tighten the upper bound of $D_{i,j}$ using the following equation:

$$\overline{D}_{i,j}[k] = \begin{cases} d_{i,j,1,1} & \text{if } \ k = 1 \\ \overline{D}_{i,j}[k-1] + \frac{\delta_n \alpha_m}{\sum_{h=1}^{h_{max}} \sum_{w=1}^{w_{max}} \delta_w \alpha_h} (d_{i,j,n,m} - d_{i,j,1,1}) & \text{if } \ k \in \{2, 3, \ldots, w_{max}h_{max}\} \end{cases} \tag{2.4}$$

where $\overline{D}_{i,j}[k]$ represents the upper bound of $D_{i,j}$ computed at the k-th iteration.

The lower bound of $D_{i,j}$ can also be gradually tightened with distances $d_{i,j,w,h}$ at finer resolutions obtained. In particular,

$$\underline{D}_{i,j}[k] = \begin{cases} d_{i,j,w^*,h^*} + \frac{\delta_1 \alpha_1}{\sum_{h=1}^{h_{max}} \sum_{w=1}^{w_{max}} \delta_w \alpha_h} (d_{i,j,1,1} - d_{i,j,w^*,h^*}) & \text{if } k = 1 \\ \underline{D}_{i,j}[k-1] + \frac{\delta_n \alpha_m}{\sum_{h=1}^{h_{max}} \sum_{w=1}^{w_{max}} \delta_w \alpha_h} (d_{i,j,n,m} - d_{i,j,w^*,h^*}) & \text{if } k \in \{2, 3, \ldots, w_{max} h_{max}\} \end{cases} \tag{2.5}$$

where $w = n$ and $h = m$ at the k-th iteration

2.2.2 Indexing and Filtering

A spatiotemporal scenario s_i is characterized by $|G| \times |T|$ number of intensity values $I_{i,n,m}$, where $I_{i,n,m}$ is the intensity at the spatial cell $g_n \in G$ and time point $t_m \in T$. For a large database S, it is very time consuming to access all $|S| \times |G| \times |T|$ records. So, in order to reduce number of scenarios to be examined, an efficient search structure is needed.

Note that the overall pairwise distance $D_{i,j}$ is lower bounded by d_{i,j,w^*,h^*}, which is a weighted difference of the total intensities and can be easily calculated if the total intensities are pre-stored in the database. Motivated by this, we create indices for all the scenarios in S using the weighted total intensity $I_i = \sum_{g_n \in G} \sum_{t_m \in T} \hat{I}_{i,n,m}$ of each scenario $s_i \in S$, where I_i is pre-calculated and stored as an independent table denoted as I. d_{i,j,w^*,h^*} can then be computed using table I, i.e., $d_{i,j,w^*,h^*} = |I_i - I_j|/|T|$ from Equation (2.3). We can then perform an initial search using d_{i,j,w^*,h^*} by only accessing table I and using the Filter-Restart concept [10].

In particular, the initial search starts with locating the scenario s_m with I_m closest to I_q, i.e., $|I_m - I_q| = \min |I_i - I_q|$, where $s_i \in S$. We then retrieve $f|S|$ number of scenarios from database that satisfy $I_i - I_q \geq I_m - I_q$ (these scenarios are saved into S_{cu}), and $f|S|$ number of scenarios from database that satisfy $I_i - I_q < I_m - I_q$ (these scenarios are saved into S_{cl}). So S_c will contain around $2 * f|S|$ number of

scenarios, i.e. $S_c = S_{cu} \cup S_{cl}$. In order to make sure that $|S_c| \geq \mathtt{K}$, we need to first check if $f|S| \geq \mathtt{K}$. In order to determine if restart is needed or not, we calculate the bounds of $D_{i,q}$ for all $s_i \in S_c$ and derive the value of $M_{\mathtt{K}}$. If all scenarios in S_{cu} (or S_{cl}) satisfy $\underline{D}_{i,q} > M_{\mathtt{K}}$, we do not need to perform the restart, otherwise, S_{cu} (or S_{cl}) is expanded by increasing the value of the threshold f.

2.2.3 Prioritizing the Windows

To further reduce the query processing time, window sizes w and h with large weights $\delta_w \alpha_h$ are prioritized. It can be done by sorting (w, h) based on weights associated to $\delta_w \alpha_h$. The resulting list of sorted window sizes is represented as $\mathcal{W} = \{(w[k], h[k])\}_{k=1}^{w_{max} h_{max}}$, where $w[k]$ and $h[k]$ represent the spatial and temporal window sizes at the k-th iteration respectively, and $\delta_{w[i]} \alpha_{h[i]} \geq \delta_{w[j]} \alpha_{h[j]}$, $\forall i < j$. Larger window sizes, indicating coarser resolutions, have smaller weights and thus less contributions to the distance calculation. Here we always have $w[1] = 1$ and $h[1] = 1$.

2.2.4 Algorithm Description

The algorithm uses the bounds of the distance measure and tighten them at each iteration to prune the search space, and find the top-K most similar scenarios. The detailed procedures of the algorithm are shown in Fig. 1. Firstly, The algorithm starts by performing the Indexing and Filtering procedure to obtain an initial candidate set S_c. After obtaining an initial candidate set S_c, this S_c set is reduced after each iteration. Specifically, at each spatiotemporal resolution, the bounds of $D_{i,q}$ for all $s_i \in S_c$ are updated, and these are used to reduce the size of the candidate set

S_c. The loop is terminated when the top `K` most similar scenarios are found or if all resolutions have been evaluated.

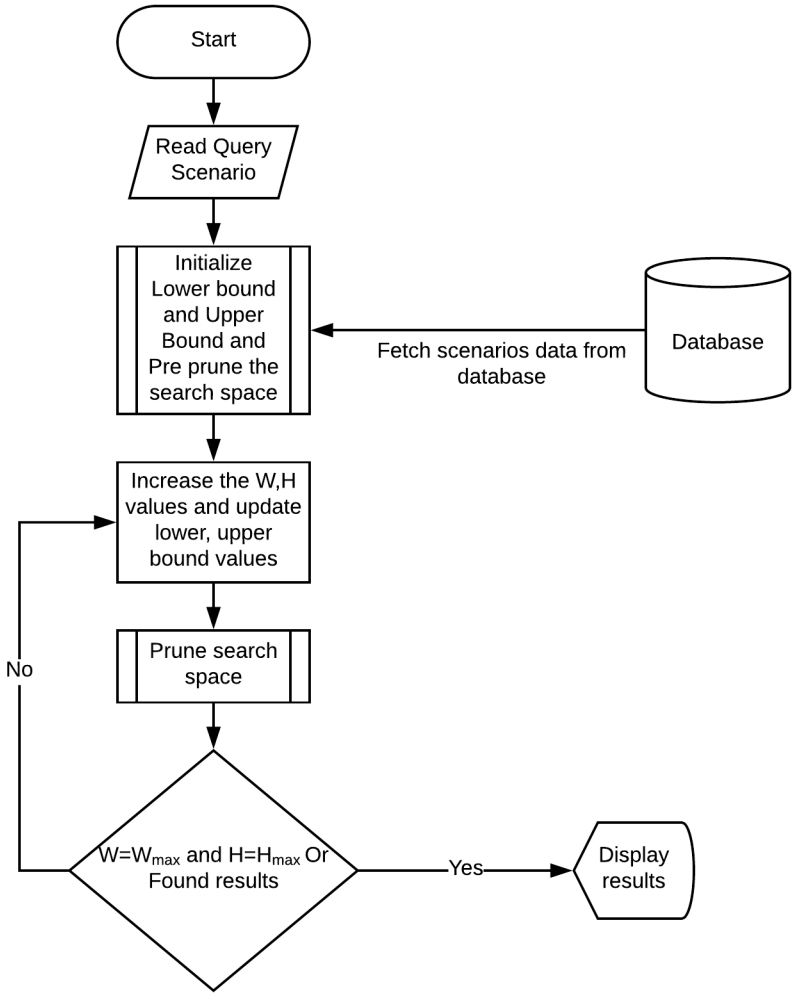

Figure 1: Algorithm procedure

CHAPTER 3

SYSTEM DESIGN

This project aims to implement the multiresolution spatiotemporal distance based similarity search algorithm and enable easy interaction between users and the database. The designed system architecture is shown in Fig.2. In particular, the application starts with a Login screen. In this screen, the user is supposed to enter database details including username, password, and database name and then click the Login button. Once the Login button is clicked, a connection is created with the database. After the database connection is established, a new screen will be displayed where the user can upload the query scenario which will be saved in an Excel file. After the query scenario is uploaded, a preprocessing is performed on the query scenario where the intensity sum for that scenario will be calculated. The user then sets the filtering criteria. On the filtering criteria is submitted, the similarity search is performed to find the top-K scenarios. Finally, the query results will display on the interface.

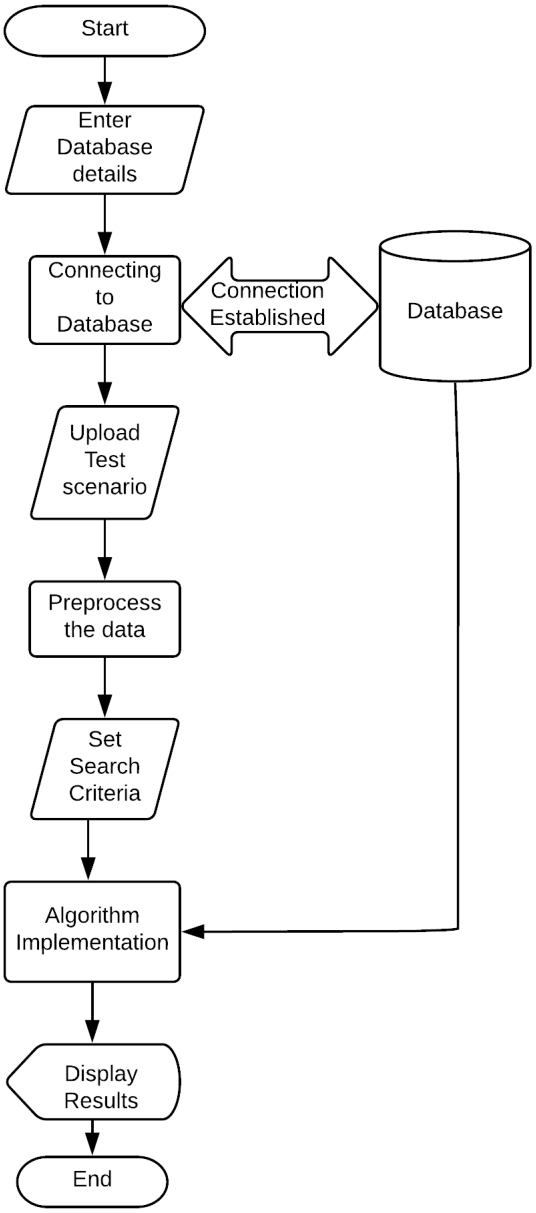

Figure 2: Proposed Framework

CHAPTER 4

SYSTEM IMPLEMENTATION

In this chapter, the system requirements and the implementation of the system architecture shown in Fig.2 is described.

4.1 System Requirements

The minimum hardware and software needed to implement the application are

Software Requirements

- Programming Languages: Java, Matlab
- Tools: NetBeans IDE 8.2 , Matlab, Java SE Development Kit 1.8.0_171
- Database: MySQL 5.7

Hardware Requirements

- Operating System: Windows 7/8/10
- RAM: 4GB
- Memory or Disk space: 100GB

4.2 Database Design

The database has 3 tables to hold the spatiotemporal scenario data.

- **Intensity Table**: This table holds the intensity values of each scenario. When scenario data is requested from this table, it returns scenario number, time, sector and intensity value.

- **Intensity Sum**: This table saves the intensity sum of each scenario. When a query is made to fetch the sum of intensities of a scenario, it returns the scenario number and scenario sum.

- **Intensity Date**: This table holds the records of scenario numbers and the scenario recorded date. When a query is made to fetch the date the scenario was recorded, it returns the scenario number and the scenario recorded date.

Other tables saved in the database include spatialImpact table, TemporalImpact table and sectindexwindow.

- **SpatialImpact Table**: This table holds the records of spatial impact value with respect to spatial cells, i.e sectors and spatial window . When a query is made to fetch the spatial impact at a particular spatial cells, then it returns the time point, temporal window and the temporal impact value.

- **TemporalImpact Table**: This table holds the records of temporal impact value with respect to time points and temporal window. When a query is made to fetch the temporal impact at a particular time point, it returns the time point, temporal window and the temporal impact value.

- **Sectorindex window**: This table holds the records of list of sectors near to a specific sector with respect to window size. When a query is made to fetch the list of sectors that can be reached in $(w - 1)$ hops, it returns the list of those sectors.

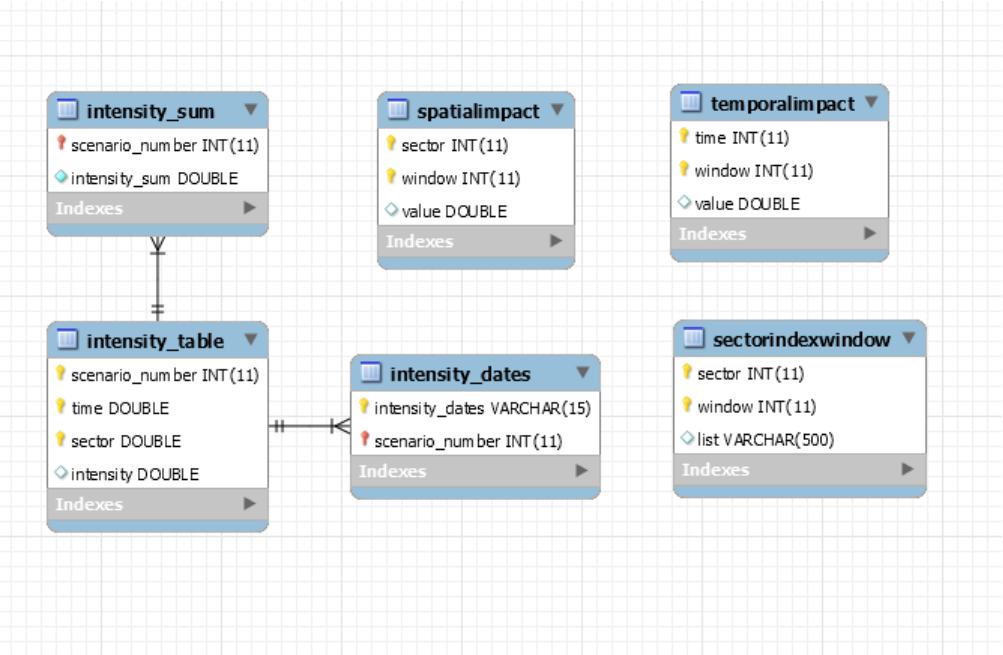

Figure 3: Entity Relation diagram of the database

Fig. 3 shows the ER diagram of the tables stored in the database.

4.3 Interface Design and Algorithm Implementation

Java SWINGS is used to implement the user interface and core Java to implement the algorithm. The external libraries as shown in Fig 4 are added into the project.

mysql-connector-java, mysql-connector libraries help the application connect to MySQL database and *poi, poi-excelant, poi-ooxml, poi-ooxml-schemas, xmlbeans, commons-collections* libraries help in reading the excel file.

The initial screen of the application is shown in Fig 6. This is the Login page to connect to the MySQL database. After entering the Username, password

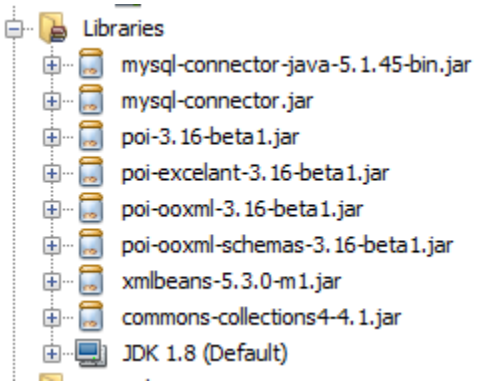

Figure 4: Screenshot of List of Java Libraries

and database name and selecting the **Login** button, if the connection is established then a database connection object is created. After creating the database connecting object , the user is allowed to upload the query scenario as shown in Fig. 8 and set the filtering criteria as shown in Fig 9. Next, we describe the functionalities of the Java application.

4.3.1 Data Insertion

The data insertion functionality is implemented using Java. In particular, after the user **Login** into the application, a button is present for the user to insert new scenarios into database. Fig 7 shows the **Insert** button. On click of **Insert** button, a new interface will appear allowing the user to insert different tables. Fig 5 shows the available options. On click of **Intensity Values** button, it opens a dialog box asking to select an Excel file to upload. After selecting the file, the application reads the Excel file, and calculates the sum of intensities of a scenario and creates three sql queries for intensity table, intensity sum, Intensity Date tables and insert the data into those particular tables.

On click of **Sector Index Window** button, user can insert the Sector indexes

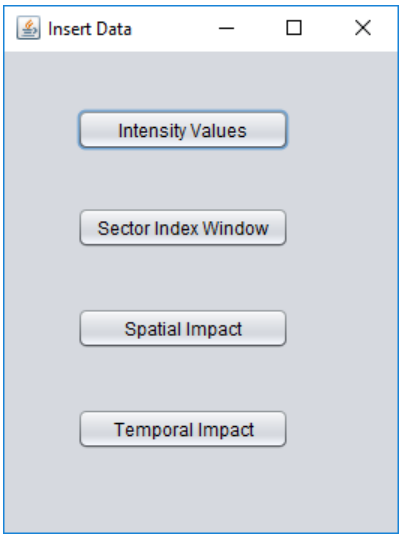

Figure 5: Options available for Inserting data

into the Sector index table as fetching data from database takes less time when compared to reading excel file on every request. As user selects **Sector Index Window** button, a dialog box appears asking to select the Sector index excel file. On selecting the excel file, the java code processes the data from excel and creates an SQL statement to insert the values into Sector Index Window table.

Similar functionalities are implemented on the **Spatial Impact** and **Temporal Impact** buttons. On click of Spatial Impact button, the uploaded excel data is stored in Spatial Impact table and on click of temporal Impact button, the uploaded excel data is stored in Temporal Impact table.

4.3.2 Filtering Criteria

After the user logs into the application, the user is allowed to set the filter criteria including the maximum spatial window size (w_{max}), maximum temporal window size (h_{max}), spatial weight coefficient (σ), temporal weight coefficient (ρ),

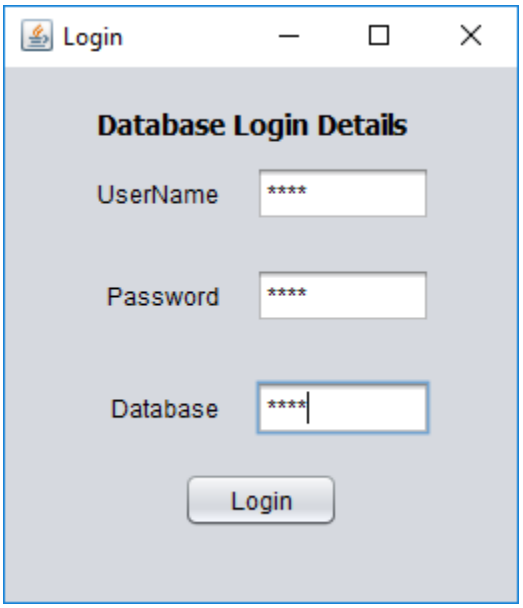

Figure 6: Login page

initial fetch percent (f) and the number of scenarios to be retrieved (K). Default values of these parameters are initially displayed. After the values of these parameters are determined, they are fetched by the algorithm, based on which the distance measure is evaluated. From fig 4 it can be seen that until the query scenario is uploaded, the **Submit** button remains disabled. On click of the **Submit** button, the application processes the query scenario, starts the similarity search, and outputs the desired top-K most similar scenarios with its details like when the scenario has been recorded. The user can also download the retrieved K scenarios by clicking the **Download Scenarios** button.

4.3.3 Similarity Search

On clicking of the **Submit** button on the user interface shown in Fig. 4, the algorithm fetches the list of all available scenarios and its intensity sum values

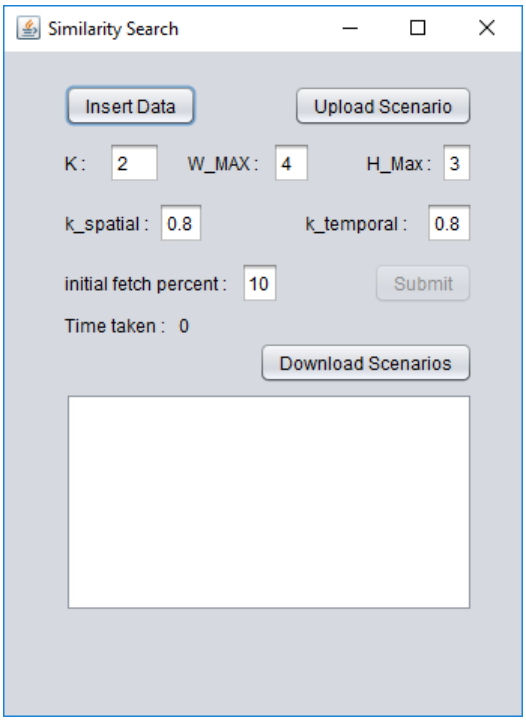

Figure 7: Similarity search initial page

from the database and performs necessary calculations to find the upper bound and perform the pre-pruning process.

In the pre-pruning step, an initial candidate set S_c is selected. Stream operations are performed in this process for higher efficiency. In particular, streams are used to create pipelines of operations which help in performing parallel computations. Threads are also used in order to make the pre-pruning process much faster. Inside threads, streams are implemented for maximizing CPU utilization.

After performing the pre-pruning, the candidate set S_c is progressively reduced to find the top-K most similar scenarios using the algorithm described in Section 2.2. The results are finally displayed on the user interface as shown in Fig. 10 application.

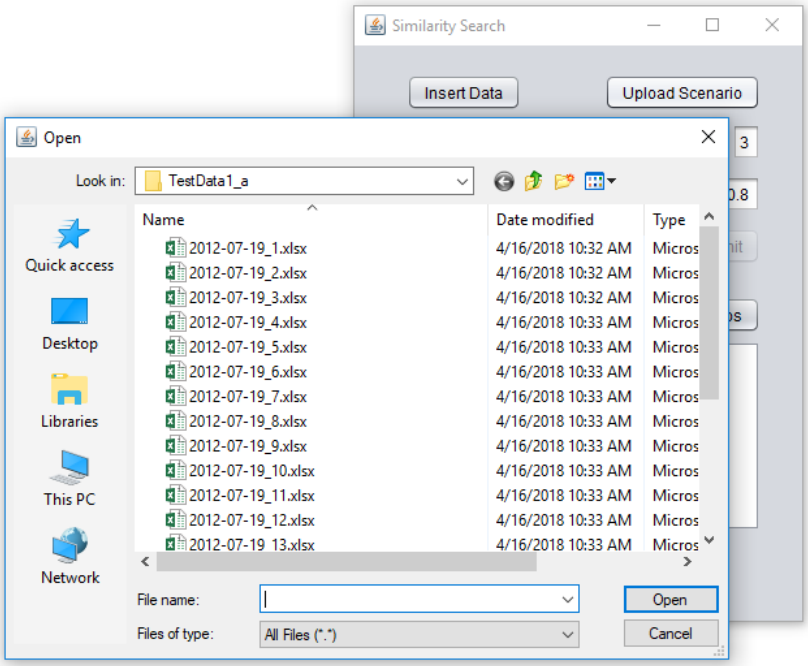

Figure 8: Selecting the input scenario

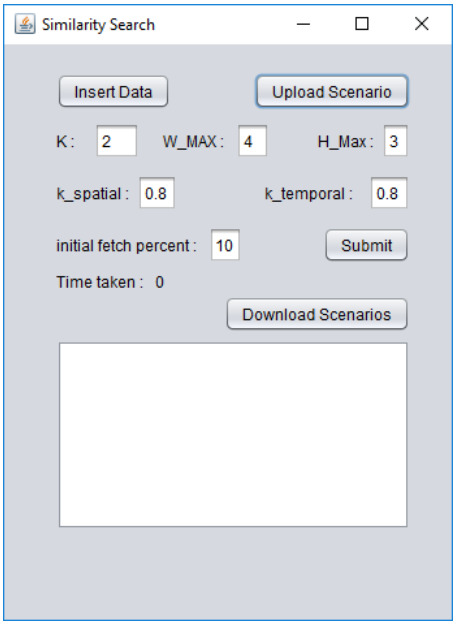

Figure 9: After uploading the input scenario

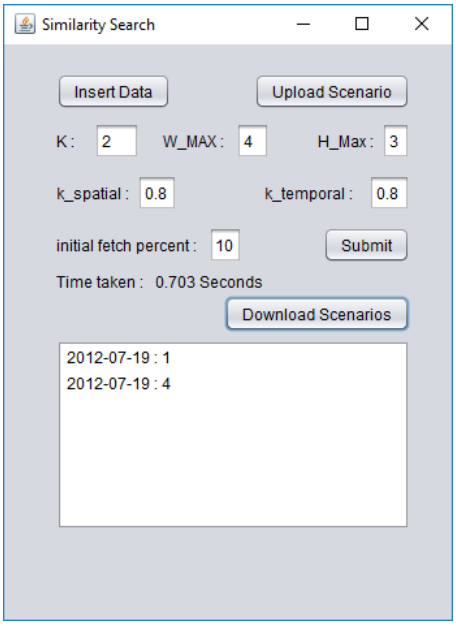

Figure 10: Results being displayed

CHAPTER 5

TESTING AND EVALUATION

In this chapter, various experiments are conducted using real weather scenarios to evaluate the performance of the similarity search algorithm implemented in Java. These experiments were run in a Dell Inspiron 17-7779 with a Intel i7-7500 processor (2.70GHz clock speed), 16GB installed RAM and 1TB hard disk.

5.1 Weather Forecast Scenarios

An ensemble forecast called short-range ensemble forecast(SREF) system [7] is used to generate the weather precipitation dataset. We have weather forecast of 62 days with each day characterized by 21 weather scenarios. Therefore, we have a total number of 1302 scenarios. Each weather scenario consists of 151 spatial cells (G) and 12 time points (T) (from 10:00Z to 21:00Z). Therefore, each scenario s_i consists of 151×12 number of records $\{g_n, t_m, I_{i,n,m}\}$, where $I_{i,n,m}$ represents the precipitation value of scenario s_i at spatial cell g_n and time point t_m. Fig 11 shows the visual representation of the weather scenario, with darker color indicating high intensity values.

5.2 Efficiency of the Java Application with increasing `K`

In this experiment, the efficiency of Java application with the increase of the query coefficient `K` is evaluated by comparing it with the Matlab prototype. The values of the parameters in the similarity search algorithm are set to $w_{max} = 4$, $h_{max} = 3$, $\delta_w = e^{-0.8(w-1)}$, $\alpha_h = e^{-0.8(h-1)}$, and $f = 0.01$. The database size is set to

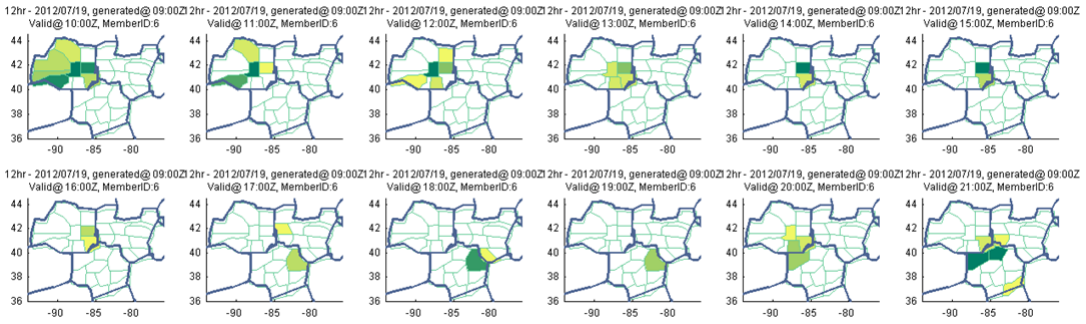

Figure 11: Visualization of an example weather scenario.

$|S| = 1302$. The comparison results are shown in Fig 12. Each value in the result is obtained by averaging the time spent to perform 21 query requests. From Fig 12, it can be inferred that the Java application takes less time when compared to Matlab prototype. We can also see that as `K` value increases, time taken to process queries also increases. This is because larger `K` leads to more number of scenarios to be evaluated at more resolutions.

5.3 Efficiency of the Java Application with increasing database sizes

In this experiment, the efficiency for accessing databases of various sizes using Java application is evaluated and compared with the Matlab application. As only 1302 actual weather scenarios are available, random scenarios are created to increase the database size to 7698, 120698, 20698, and 30698. The values of the parameters in the similarity search algorithm are set to $w_{max} = 4$, $h_{max} = 3$, $\delta_w = e^{-0.8(w-1)}$, $\alpha_h = e^{-0.8(h-1)}$, $\mathtt{K} = 2$, and $f = 0.01$. The comparison results are shown in Fig 13. Each value in the result is obtained by averaging the time spent to perform 21 query requests. The results further demonstrate the efficiency of the Java application. In addition, the execution time of the Java application increases with the increase of

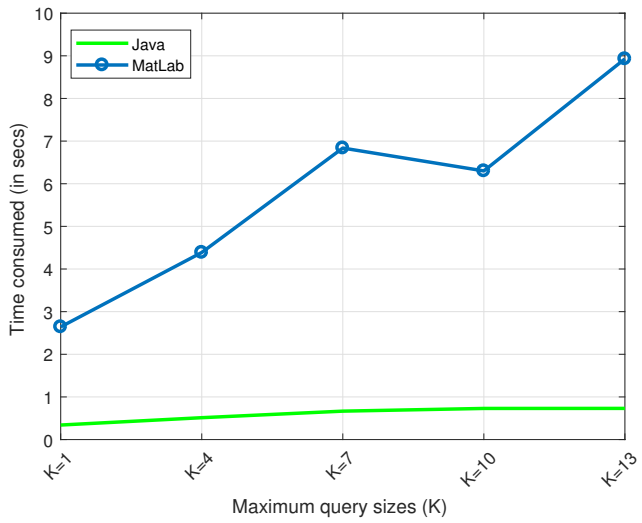

Figure 12: Execution time versus the K

the database size. This is because of the increase of the size of the candidate set (S_c). However, in Matlab, the time consumed to process arithmetic operations is relatively low when compared to Java. Because of that, as we can see, though database size increases, the time consumed to process the query remains almost constant. The large execution time at small database sizes is caused by the frequent restarts to extract an initial candidate set of proper size.

5.4 Efficiency of the Java Application with increasing threshold f

In this experiment, the efficiency of the Java application with the increase of the threshold f that determines the size of the candidate set S_c is evaluated and compared with the Matlab application. The values of the parameters in the similarity search algorithm are set to $w_{max} = 4$, $h_{max} = 3$, $\delta_w = e^{-0.8(w-1)}$, $\alpha_h = e^{-0.8(h-1)}$, $\mathtt{K} = 13$, and $|S| = 1302$. The comparison results are shown Fig 14. Each value in the result is also obtained by averaging the time spent to perform 21 query requests. As

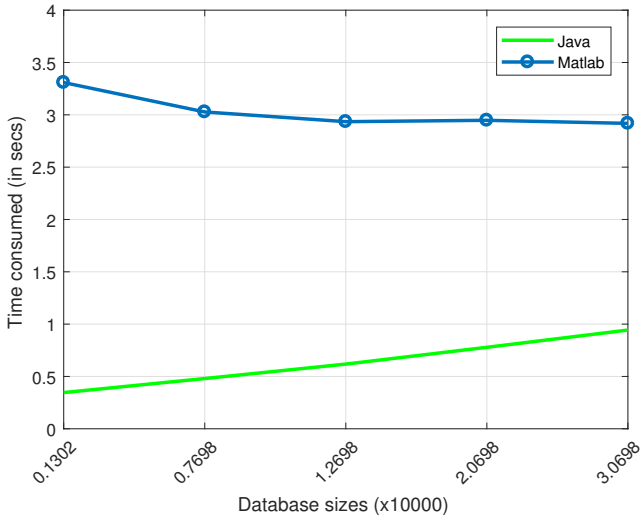

Figure 13: Execution time versus the database size

shown in Fig 14, Java application is more efficient compared to the Matlab prototype. We can also see that as f value increases, time taken to process queries also increases. This is because, as f value increases there would be a larger number of unnecessary scenarios in the candidate set(S_c).

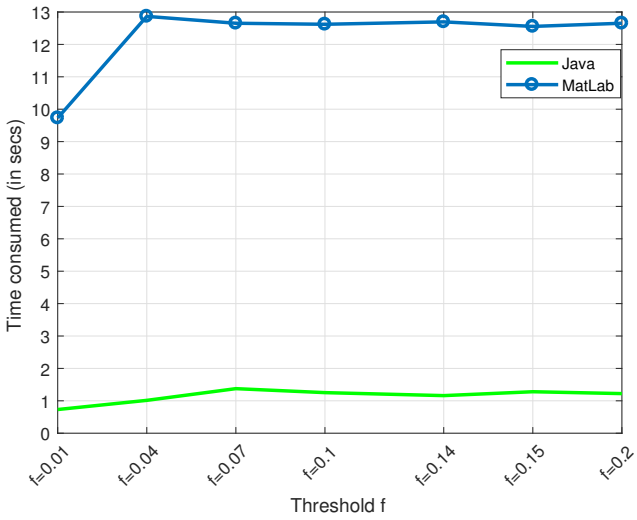

Figure 14: Execution time versus the threshold f

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this work, a similarity search algorithm for spatiotemporal scenario data has been implemented using Java as framework and a comparative study is conducted with Matlab. The main purpose of this work is to create a platform independent application which can run anywhere without any constrains and produce efficient results with high efficiency. The results show that the present implementation is much more efficient than the Matlab-based prototype. As part of future work, the database can be hosted on the Internet so that users will be able to run the application from anywhere in the world.

REFERENCES

[1] Berndt, D. J., and Clifford, J. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining* (1994), AAAIWS'94, AAAI Press, pp. 359–370.

[2] Birant, D., and Kut, A. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering 60*, 1 (2007), 208–221.

[3] Bruno, N., Gravano, L., and Marian, A. Evaluating top-k queries over web-accessible databases. In *Data Engineering, 2002. Proceedings. 18th International Conference on* (2002), IEEE, pp. 369–380.

[4] Cao, P., and Wang, Z. Efficient top-k query calculation in distributed networks. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing* (2004), ACM, pp. 206–215.

[5] Chang, K. C.-C., and Hwang, S.-w. Minimal probing: supporting expensive predicates for top-k queries. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data* (2002), ACM, pp. 346–357.

[6] Chen, L., Özsu, M. T., and Oria, V. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (2005), ACM, pp. 491–502.

[7] Du, J., DiMego, G., Zhou, B., Jovic, D., Ferrier, B., Yang, B., and Benjamin, S. Ncep regional ensembles: Evolving toward hourly-updated convection-allowing scale and storm-scale predictions within a unified regional

modeling system. In *22nd Conf. on Numerical Weather Prediction/26th Conf. on Weather Analysis and Forecasting* (2014).

[8] Fagin, R., Lotem, A., and Naor, M. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences 66*, 4 (2003), 614–656.

[9] Guntzer, J., Balke, W.-T., and Kiessling, W. Towards efficient multi-feature queries in heterogeneous environments. In *Information Technology: Coding and Computing, 2001. Proceedings. International Conference on* (2001), IEEE, pp. 622–628.

[10] Ilyas, I. F., Beskales, G., and Soliman, M. A. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR) 40*, 4 (2008), 11.

[11] Jeung, H., Yiu, M. L., Zhou, X., Jensen, C. S., and Shen, H. T. Discovery of convoys in trajectory databases. *Proceedings of the VLDB Endowment 1*, 1 (2008), 1068–1080.

[12] Li, C., Chang, K. C.-C., Ilyas, I. F., and Song, S. Ranksql: query algebra and optimization for relational top-k queries. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* (2005), ACM, pp. 131–142.

[13] Li, C., Chen-Chuan Chang, K., and Ilyas, I. F. Supporting ad-hoc ranking aggregates. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (2006), ACM, pp. 61–72.

[14] Lin, B., and Su, J. Shapes based trajectory queries for moving objects. In *Proceedings of the 13th annual ACM international workshop on Geographic*

information systems (2005), ACM, pp. 21–30.

[15] Mokbel, M. F., Xiong, X., and Aref, W. G. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (2004), ACM, pp. 623–634.

[16] Ohashi, H., Shimizu, T., and Yoshikawa, M. Flexible similarity search for enriched trajectories. In *Data Mining Workshops (ICDMW), 2016 IEEE 16th International Conference on* (2016), IEEE, pp. 1139–1144.

[17] Pripužić, K., Žarko, I. P., and Aberer, K. Time-and space-efficient sliding window top-k query processing. *ACM Transactions on Database Systems (TODS) 40*, 1 (2015), 1.

[18] Sakurai, Y., Yoshikawa, M., and Faloutsos, C. Ftw: fast similarity search under the time warping distance. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2005), ACM, pp. 326–337.

[19] Soliman, M. A., Ilyas, I. F., and Chang, K. C.-C. Top-k query processing in uncertain databases. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on* (2007), IEEE, pp. 896–905.

[20] Tork, H. F. Spatio-temporal clustering methods classification. In *Doctoral Symposium on Informatics Engineering* (2012), pp. 199–209.

[21] Vlachos, M., Kollios, G., and Gunopulos, D. Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on* (2002), IEEE, pp. 673–684.

[22] Wang, X., Shen, D., and Yu, G. Uncertain top-k query processing in distributed environments. *Distributed and Parallel Databases 34*, 4 (2016), 567–589.

[23] Xie, J., Nguyen, H., and Wan, Y. Similarity search of spatiotemporal scenarios for strategic air traffic management. In *AIAA Aviation 2018 (accepted)* (2018).

[24] Xie, J., Wan, Y., Zhou, Y., Tien, S.-L., Vargo, E. P., Taylor, C., and Wanke, C. Distance measure to cluster spatiotemporal scenarios for strategic air traffic management. *Journal of Aerospace Information Systems 12*, 8 (2015), 545–563.

[25] Xie, J., Zhou, Y., Wan, Y., Tien, S.-L., Taylor, C. P., and Wanke, C. R. A multi-resolution spatiotemporal scenario clustering algorithm for flow contingency management. In *14th AIAA Aviation Technology, Integration, and Operations Conference* (2014), p. 2029.

[26] Zeinalipour-Yazti, D., Lin, S., and Gunopulos, D. Distributed spatio-temporal similarity search. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 2006), CIKM '06, ACM, pp. 14–23.

[27] Zhang, Z., Hwang, S.-w., Chang, K. C.-C., Wang, M., Lang, C. A., and Chang, Y.-c. Boolean+ ranking: querying a database by k-constrained optimization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data* (2006), ACM, pp. 359–370.

APPENDIX A

CODE SNIPPETS OF THE PROJECT

The following code is for connect to database

```
public dbConnection( String username, String password, String databaseName ) {
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        return;
    }
    try {
        connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/"
        +databaseName,username,password);
    } catch (SQLException e) {
        System.out.println("Connection Failed! Check output console");
        return;
    }
    if (connection != null) {
        System.out.println("You made it, take control your database now!");
    } else {
        System.out.println("Failed to make connection!");
    }
}
```

The following code is to call a function from login page to connect to database

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    userName = username.getText();
    passWord = password.getText();
    databaseName = database.getText();
    if (!userName.equals("") && !passWord.equals("")
     && !databaseName.equals("")) {
        dbConnection conn =
         new dbConnection(userName, passWord, databaseName);
        if (conn != null) {
            new Options(conn).setVisible(true);
            setVisible(false);
        } else {

        }
    }
}
```

The following code is to upload test scenario from local machine to JAVA code

```
void UploadScenario(String file) {
    try {
        FileInputStream inputStream =
         new FileInputStream(new File(file));
        Workbook workbook = new XSSFWorkbook(inputStream);
        Sheet firstSheet = workbook.getSheetAt(0);
        Iterator<Row> iterator = firstSheet.iterator();
        dm1 = 0d;
        double val;
        String str1;
        S1.clear();
        while (iterator.hasNext()) {
            Row nextRow = iterator.next();
            Iterator<Cell> cellIterator = nextRow.cellIterator();
            while (cellIterator.hasNext()) {
                Cell cell = cellIterator.next();
                cell = cellIterator.next();
                str1 = cell.toString();
                val = Double.parseDouble(cell.toString());;
                S1.add(val);
            }
        }
        IntStream.range(0, S1.size())
            .forEach(j -> {
                dm1 = dm1 + S1.get(j);
            });
    } catch (FileNotFoundException ex) {
        Logger.getLogger(FindSimilar.class.getName())
         .log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(FindSimilar.class.getName())
         .log(Level.SEVERE, null, ex);
    }
}
```

Code to fetch scenario intensity values from database

```
public List<Double> getData(int index) {
    List<Double> s2 = new ArrayList<>();
    try {
        Connection connection = conn.getConnection();
        Statement stmt = connection.createStatement();
        String query = "SELECT * FROM intensity_table where
            scenario_number =" + index + ";";
        ResultSet rs = stmt.executeQuery(query);
        int count = 0;
```

```
        int time = 10;
        int t = 0;
        while (rs.next()) {
            t = 151 * timeIndex.get(rs.getInt("time"))
                        + rs.getInt("sector");
            if (t != (count + 1)) {
                for (int i = (count + 1); i < t; i++, count++) {
                    s2.add(0d);
                }
            }
            s2.add(rs.getDouble("intensity"));
            count++;
        }
        if(s2.size() < 1812) {
            for(int i=s2.size();i<1812;i++) {
                s2.add(0d);
            }
        }
        stmt.close();
    } catch (SQLException ex) {
        Logger.getLogger(FindSimilar.class.getName())
         .log(Level.SEVERE, null, ex);
    }
    return s2;
}
```

The following code is to fetch scenario details from database inorder to display final result

```
private List<String> fetchScenarioDetails(List<Integer> M) {
    List<String> scenarioDates = new ArrayList<>();
    try {    Connection connection = conn.getConnection();
        try (Statement stmt = connection.createStatement()) {
            String query = "SELECT * FROM intensity_dates where ";
            for(int i=0; i<M.size(); i++) {
                if( i > 0 ) {   query = query + " or ";       }
                query = query + "scenario_number = "+M.get(i);
            }
            ResultSet rs = stmt.executeQuery(query);
            while (rs.next()) {
                scenarioDates.add(rs.getString("intensity_dates")+" :
                    "+rs.getInt("scenario_number"));
            }
        }
    } catch (SQLException ex) {
        Logger.getLogger(FindSimilar.class.getName())
        .log(Level.SEVERE, null, ex);    }
    return scenarioDates;
}
```