Final Year Project Interim Report

SpaceKey: Exploring Patterns in Spatial Databases

Supervisor: Dr. Reynold C. K. Cheng

Member:
Yin Yue / 3035234234
Wang Jikun / 3035234715
Lukito Budiman / 3035253864
Abstract

With the significant market size of real-estate market across the world, the demand for property has reached a new level. At the same time, searching for properties through the internet has become a convenient approach for users. Existing web-based properties searching applications such as Airbnb and 28hse.com only support simple searching and filtering functionalities. However, in real life scenarios, users might have more complex requirements, for example, the user might want an mtr station not too far from his property. With the help of recent researches on spatial group keyword queries, the development of this kind of utility is made possible. This project is set to demonstrate the feasibility and importance of this advanced searching utility by developing a standalone web-based application. In the project, we want to utilize commercial map data sources such as Google maps API and OpenStreetMap and use Spatial Pattern Matching (SPM) as our primary model. Currently the project is half way through its development process, and a fully functional demo in terms of basic functionalities is upon completion. This report will introduce the project and the progress of the project in detail by the date of submission.

Acknowledgement

I would like to express my appreciation to all the people who have provided supports during the process of completing the report, especially our project supervisor, Dr. Reynold C. K. Cheng. His contributions helped the project a lot in terms of giving suggestions and encouragement, as well as coordinating the project.
**Table of Content**

Abstract 1
Acknowledgement 1
Table of Content 2

1. Project Background 3

2. Project Objectives 4
   2.1 Supporting APIs 4
   2.2 Sample dataset 4
   2.3 A standalone demo housing application 4
   2.4 A project website 5

3. Project Methodology 6
   3.1 Data collection 6
      3.1.1 PoI data 6
      3.1.2 Property data 6
      3.1.3 Data storage 6
   3.2 Application of SPM 7
      3.2.1 The problem of unwanted object 7
      3.2.2 The problem of realistic distance 7
      3.2.3 Directional relations 7
      3.2.4 Custom objects 7
      3.2.5 Additional attributes 7
   3.3 Development structure and specs 8

4. Project Progress 9
   4.1 Data collection 9
      4.1.1 Crawler implementation 9
      4.1.2 Difficulty 9
      4.1.3 Data volume 9
   4.2 Application of SPM 9
      4.2.1 The API 10
      4.2.2 The problem of unwanted object 10
      4.2.3 Directional relations 11
      4.2.4 The problem of realistic distance 11
      4.2.5 Custom objects 11
   4.3 Website application 11

5. Future Plan 13

References 14
1. Project Background

The demand for property has reached a new level with the significant market size of real-estate market across the world [1]. At the same time, searching for properties through the internet has become a convenient approach for users. There are several existing properties searching applications, such as Airbnb and 28hse.com, which help users to find their desired property locations. After investigating on these properties searching applications, observations show that these applications only support simple searching functionalities, by searching for single elements and applying simple filters\(^1\). However, in real life scenarios, users might have more complex requirements on their desired property, for example, the user might want a school as well as a hospital not too close nor too far from his property. As a result, a more advanced searching utility is in demand.

Recently, a new type of spatial group keyword query called Spatial Pattern Matching (SPM) [2] is raised in research. Spatial pattern is a graph with its vertices being keywords and its edges being distance constraints between any pair of keywords. This type of query matches a group of objects with their keywords and distance constraints satisfying the spatial pattern specified by the user. Its solutions have been raised and well-discussed. With the help of SPM and its solutions, it is possible to achieve the advanced searching functionality mentioned above. As a result, our outlook is to realize this kind of advanced searching utility in these housing applications as a module. In this project, our goal is to demonstrate the feasibility and importance of this searching utility by developing a standalone web-based application.

The remaining parts of this report were arranged as the followings. First, we will describe the four main objectives of our project in detail, which are the building a set of supporting APIs, a sample dataset, the main application as well as the project website. After that, the project methodology is described. Then the current progress of the project by the date of submission is discussed, which includes our work on data collection, building the API and developing the website, as well as the difficulties and challenges encountered during this period. Then, the planning of future development according to the current progress will be presented, which mainly focuses on the development of the web application and innovative features and functionalities. Finally, a conclusion will be drawn for this progress report.

\(^1\) Simple filters here are defined as filters that applied using simple inputs and only applied to the property itself. For example, a filter based on price range.
2. Project Objectives

This project’s outlook is to integrate a module with existing housing applications to do advanced house searching with the help of SPM. In this module, users can filter available housing locations according to their interest in surrounding keyworded Points of Interests (PoI), such as supermarkets and schools. They can specify distance and directional constraints between the desired property and PoIs or user-specified locations. At the same time, it will provide more location-related information to the user, such as 24-hour traffic, reviews, ratings, etc. In this project, the main goal is to make a standalone demo application that shows how the advanced house searching works, which includes all the above functionalities.

2.1 Supporting APIs

The first objective is to build a set of open-source supporting APIs for the project. It will focus on the spatial query algorithms, specifically SPM, as well as many other existing algorithms, such as top-k [3, 4], m-Closest Keywords (mCK) [5], minimum Spatial Keyword cover (minSK) [6], Collective Spatial Keyword Queries (CoSKQ) [7, 8]. It will achieve following basic functionalities:

- Input data from local sources;
- Build various data structures to support corresponding types of spatial keyword queries
- Answer various spatial keyword queries

The API set will be open-source so anyone can use it to develop new applications in the future.

2.2 Sample dataset

The next objective is to obtain a sample dataset for the demo application. As required by the SPM algorithm, the dataset needs to include the following information:

- Housing locations:
  - Spatial Information: latitude, longitude;
  - Housing Information: price, house size, etc.
- Points of Interest (PoI) locations:
  - Spatial Information: latitude, longitude;
  - Keyword Information: A set of feature keywords for each object;
  - Additional Information: ratings, reviews, etc.
- Distance information:
  - Realistic distance: travel time, 24-hour traffic, etc.

It should be noted that the sample data could either be authentic data or generated data, depending on our approach of making the demo application.

2.3 A standalone demo housing application

A standalone demo housing application will be developed to demonstrate the industrial value of SPM and realize its outlook. The demo will be a web-based application. It will have following basic functionalities:

- Pattern Specification: the demo application will feature an easy-to-use and advanced pattern input interface. They can also specify distance and direction requirements. It will also allow users to specify other requirements for the location, such as rush-hour traffic. In
addition, other functionalities and options will be provided, such as inputting custom objects.

- Query processing: The demo application will be able to process the query utilizing the supporting API mentioned above and return the results in a considerably short amount of time.
- Results display: the demo application will feature a highly comprehensible result display based on a map. It will display all the results that satisfy the requirements from the users. Additional information regarding the locations will also be displayed if needed, such as traffic information, reviews, and ratings.

2.4 A project website

A website will be developed to display information related to the project. It will include the following contents:

- The detail information of this project, including the objective and project’s progress;
- The documentation of the usage of the supporting API set;
- The documentation and tutorial of the usage of the demo housing web application;
- Files related to this project, such as project’s plan, interim report, and final report;
- Some other materials related to the project, such as the libraries used.
3. Project Methodology

3.1 Data collection

Data collection is the first step of our project. Determined by the goal of our project, there are two types of data that we need to collect - PoI data and Property data. Each type of data needs to be obtained from different data sources using different techniques. The following sections will be discussing both scenarios as well as how we store these data.

3.1.1 PoI data

PoI data is collected from Google Maps website by crawling. Python libraries such as Selenium are utilized to simulate interactions by controlling the browser through webdrivers. This crawling process starts from initiating a search on Google Maps with a manually chosen keyword (e.g. School, Hospital) on the webdriver. Then the plus code\(^2\) in the result is passed to another webdriver for translation into coordinates using Google Maps. Finally, the result collected is formatted for storage in the database. Chrome webdriver is currently adapted and set to ‘headless mode’ as well as ‘disable GPU’ so as to avoid additional memory allocation and improve efficiency. Routing data between PoIs can also be collected through similar procedure.

3.1.2 Property data

Property data is gathered from 28hse.com and Airbnb, where 28hse.com provides precise coordinates of the locations as well as other information such as contact person. On the other hand, Airbnb only provides approximate location to protect privacy and it has no direct contact information. The procedure follows the structure of our Google Maps crawler with the same technical stack, while the result page exploration is based on URL. The URL of the property will also be stored along with the property specifications (e.g. price, area, region) for redirecting links and uniqueness validation.

3.1.3 Data storage

Data formats are defined as database tables with null value as default for illegal and missing entries. The uniqueness of a PoI data entry is validated by the combination of name and coordinates. On the other hand, the uniqueness of property is determined by the redirecting URL. The old data record will be replaced when inserting a duplicate record to update the entry. The database instance is deployed on AWS RDS\(^3\), which is connected through mysql.connect during record insertion. Most database operations are wrapped to be asynchronous to prevent itself from blocking the main crawling procedure.

All the data stored in database will be the domain of the SPM algorithm, so it is exposed to the API server for data access. At the same time, the crawler will run periodically to ensure the data is up-to-date.

---

\(^2\) Plus code is a combination of shortcode and locality which describe a unique location. For instance, 9CG+2X Noida.

\(^3\) AWS RDS is Amazon Relational Database Service, a web service that makes it easier to set up, operate, and scale a relational database in the cloud.
3.2 Application of SPM

In this project, SPM is our major problem model of our application. In order to apply SPM to our project, we need to fit the theory to the real scenario. The following sections will be discussing some challenges when directly applying the SPM problem model to our project, as well as some ideas to make better use of the algorithms.

3.2.1 The problem of unwanted object

In SPM, when a keyword is specified in the pattern, the returned result must contain that keyword regardless of the magnitude of the distance constraints [2]. In the real world scenario, however, when a user just want any instance of a keyword to be far enough from the location such as airport, the user will not care about any instance of that keyword. Furthermore, matching an instance that is far away will hurt the performance of the query. Therefore, we want to propose some approaches to solve this problem and get acceptable performance.

3.2.2 The problem of realistic distance

In SPM, the distance is measured by euclidean distance between locations, which is not very realistic: locations that are geometrically closer to each other might take more time to travel between. Therefore more realistic distance measurement is needed in this application, such as travel time and route distance. Since the algorithm requires distance data to be local, we need to obtain routing data instead of invoking existing APIs, such as Google Maps API. Due to the routing data source limitations mentioned above, in the case of no other data source is available, we propose alternative ways to utilize these data: Euclidean distance will be used in the algorithm, with looser distance intervals decided by the app’s specification. Then, the routing queries on a map API will be used to get the realistic distance only on the result set to filter/display them afterwards.

3.2.3 Directional relations

In SPM, no directional relations could be specified in a pattern. However, in a real world scenario, a user might want to specify directional constraints between objects, e.g. a school to be on the northwest of the housing location. This functionality can be achieved by applying a filter on the result set to validate directional relations.

3.2.4 Custom objects

We want to enable user with the ability of specifying custom objects in the pattern. It could be useful if the user has some locations in the user’s mind to relate to, for instance workplace and friends’ house. In our project, the user will be able to specify a custom object by clicking on the map or specify latitude and longitude. We can achieve this by assigning a unique keyword to the location that the user specifies temporarily, so that the result must include the location with the that keyword. In addition, more useful information could be displayed related to the custom objects. In the case of workplace or friends’ house, 24-hour traffic information (especially rush hours) will be very useful for the user. This information could be obtained from various existing map APIs.

3.2.5 Additional attributes

In our demo, we want to add the functionality to display other information related to the housing location and PoIs, which might possibly helpful for the user to find the desired property. The additional attributes include ratings, the number of reviews, user review, opening hours, etc. Also,
we can use some attributes to decide the displaying order of the results, as well as filtering them, such as ratings and the number of reviews. These data can be pulled from various existing map APIs.

### 3.3 Development structure and specs

**API set:**

The supporting API set is developed using Java. It will be later hosted either on a cloud service such as AWS or on our own dedicated machine.

**Database:**

The database is developed and maintained in MySQL. It will be later hosted on a cloud service such as AWS or on our own dedicated machine.

**Web application:**

For the front-end, the web-application will be built with React.js framework and Semantic UI React as its UI library. For the back-end, Node.js will be used.
4. Project Progress

4.1 Data collection

This section will cover current progress of data collection and the difficulty encountered.

4.1.1 Crawler implementation

Data crawlers for PoI and property data are implemented through a similar procedure. During the development of the data crawler, the common technique, which relies on parsing AJAX response, is abandoned. The reason is that the AJAX response from Google Maps is an encoded URL pointing to a txt file storing requested data. However, the returned data is filled with enormous null values and brackets, which is hard to parse without any knowledge about the protocol behind. Therefore, parsing from html content is adapted. All crawlers are completed with exception handlers to replace illegal entries with ‘null’ and keep the crawler from accidental shutdown. Crawlers are deployed on AWS EC2\(^4\) server instances to achieve automation and concurrency.

4.1.2 Difficulty

The crawling process of Google Maps begins with a starting point which is passed as a parameter in the request. Since the range of each search is limited by the distance from the starting point, in order to gather the data from the whole region, it is required to initiate search from multiple starting points across the map. Therefore, we initiate our queries from random starting points chosen from the whole map. With uniqueness check while inserting data, there will be no duplicates in our dataset. After a number of runs, most of the PoI data will be collected.

As mentioned in 3.1.1, the routing data follows the same procedure. However, the size of the routing table is proportional to the square of the data size. As the data set grows, it becomes more difficult to crawl the routing data. It may be achieved if proxy servers are integrated into crawler, therefore the feasibility requires further investigation.

4.1.3 Data volume

According to the database record count, the collected data volume is about 8000 entries of PoIs from Google Maps. Additionally, there are about 17,000 entries of properties for sell and about 23,000 entries of properties for rent from 28hse.com and nearly 10,000 entries from Airbnb for short term rent. With our automated crawlers, the PoI data size is expected to grow and the property data will be updated.

4.2 Application of SPM

In this section, we will discuss the progress of the application of SPM, including the development progress of the API, as well as how we have attempted to address the ideas and challenges previously mentioned.

\(^4\) AWS EC2 is Amazon Elastic Compute Cloud, a web service that provides secure, resizable compute capacity in the cloud.
4.2.1 The API

The API is currently ready to deploy and already up and running in the LAN environment. The following functionalities are already tested and communicates well with the website:

- Returning property data with filters on type (buy/rent) and region as request parameters;
- Returning Pol data with filters using wanted Pols as request parameters;
- Returning SPM queries with simple request parameters, as shown in the simple SPM query of the website in section 4.3.

The following functionalities are already tested in console but not yet implemented for the website:

- Specify directional relations when configuring a SPM query;
- Setting a user-defined custom object in the map;
- Returning SPM queries with complex request parameters (the whole spatial pattern).

4.2.2 The problem of unwanted object

Upon this challenge, we define the problem and propose three potential approaches to deal with this problem.

Suppose the query pattern is $P$, the set of unwanted keywords is $U$.

1. Directly use the SPM algorithm on $P$, setting the interval upper-bound with the unwanted object to a large number ($+\infty$);

2. Use the SPM on $P \setminus U$, and filter the result set so that no unwanted keywords in $U$ is within the unacceptable distance of the corresponding keywords;

3. Use the SPM on $P \setminus U$, get result set $R$. Then for each keyword $u_i$ in $U$, perform SPM on $P - U + \{u_i\}$, get result set $R_i$. Then the final result will be $R - \text{SUM}\{R_i\}$.

Among these three approaches we proposed, the first one needs to have a substance of the unwanted keyword to have a result, therefore in the case of non-existent keyword on the map, it could be a problem. For example, when a user does not want an airport close to his property and there is actually no airport in the map. In this case, the approach 1 will return nothing, which does not satisfy the user’s request.

The approach 2 and 3 eliminate this problem because instead of directly using the SPM algorithm with $P$, they use $P \setminus U$, which excludes the unwanted keywords, and filter the result sets afterwards.

Looking at approach 2 and 3, since only the filtering techniques are different - the approach 2 is filtering results by looking at unwanted keywords, while the approach 3 does it by performing SPM on result set with every additional unwanted keyword. Therefore, the meaning behind these two approaches are equivalent. In addition, if all the unwanted keywords exist on the map, the approach 1, 2, and 3 have the same problem to solve, therefore they are equivalent in this case.

In terms of efficiency, we have tested all three in the UK PoI data with keywords (since our large-scale database is not ready yet), the second approach is better than first approach and far better than the third one. The first one is slow potentially because the specification of the long distance constraint extends the searching area by too much. The third one is slow potentially because of the huge overhead of performing too many SPMs.

As for now, we opt for the second approach in our project. Whenever there are new ideas available, we will investigate them and adjust our approach accordingly.
4.2.3 Directional relations

As discussed earlier, this functionality can be achieved by adding another layer of filter on the result set to validate if they satisfy the directional relations. The idea has been already implemented and deployed on our website.

4.2.4 The problem of realistic distance

Regarding our proposal of using euclidean distance to query and routing API to filter out hard-to-access objects, it is ready to deploy upon the completion of the website. However, we are still attempting to crawl the data from some available routing websites such as Google Maps and OpenStreetMap, but its feasibility is yet to be proven, which is shown in section 4.1.2.

4.2.5 Custom objects

This idea is already addressed and developed in the API. When a user specify a custom object, the system will temporarily generate an object with a unique keyword for this user. When the user specify a spatial pattern with the custom object, the unique keyword will be used to match that object. Related functionalities which are mentioned in section 3.2.5, such as displaying useful information with regard to the custom object, will be added upon deployment.

4.3 Website application
At this moment, a property-search page has been developed partially. As shown in figure 1, the search page includes a search bar to specify the region and two buttons to specify the property type (rent or buy). After users input the region in the search bar, the website will send a HTTP request to the API server to obtain the property data. In addition, the search page also includes some simple filters such as number of bedrooms and price. These filters are implemented on the client-side to reduce the numbers of HTTP requests needed. Furthermore, the user can also specify distance constraints to the wanted PoIs, as shown in figure 2. After user sends the query, the result will be shown both as a list and as pin points on the map.
5. Future Plan

In the beginning of the semester 1, we planned to make the mobile version as well. However, after the interim presentation, we decided to focus more on how to utilize SPM since our main goal is to explore real-life challenges on the application of the theory.

For the time being, our focus will be on collecting the routing data from online data resources to tackle the problem addressed in section 4.2.4, the realistic distance problem. In addition, we will implement the functionalities mentioned in section 4.2.1.

The detailed schedule for the second semester is presented in the table below.

<table>
<thead>
<tr>
<th>Date</th>
<th>Milestones</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-31 January</td>
<td>● Collecting routing data</td>
</tr>
<tr>
<td></td>
<td>● Implementing the API features on section 4.2</td>
</tr>
<tr>
<td>1-28 February</td>
<td>● API and website deployment into a dedicated server</td>
</tr>
<tr>
<td></td>
<td>● Auto scheduled data crawler</td>
</tr>
<tr>
<td></td>
<td>● Coming up with and testing new ideas for the application of the algorithms</td>
</tr>
<tr>
<td>1-31 March</td>
<td>● UAT testing &amp; optimization on the website and APIs</td>
</tr>
<tr>
<td></td>
<td>● Coming up with and testing new ideas for the application of the algorithms</td>
</tr>
<tr>
<td>1-13 April</td>
<td>● Final testing</td>
</tr>
<tr>
<td></td>
<td>● Preparing Final report, final presentation, and project exhibition</td>
</tr>
<tr>
<td>14 April</td>
<td>Deliverable of Phase 3 (Construction)</td>
</tr>
<tr>
<td></td>
<td>● Finalizing tested implementation</td>
</tr>
<tr>
<td></td>
<td>● Final report</td>
</tr>
<tr>
<td>15-19 April</td>
<td>Final presentation</td>
</tr>
<tr>
<td>29 April</td>
<td>Project exhibition</td>
</tr>
</tbody>
</table>

6. Conclusion

This progress report introduces our project, SpaceKey, which is an advanced property-searching application. The project has several main objectives, which includes the acquisition of sample data, the development of supporting APIs, as well as the development of the final standalone application. Current progress of the project, including the study of previous researches and approaches to acquire data, shows the feasibility of some features, as well as some difficulties and limitations of future development, especially on data acquisition. For example, the efficiency is limited while crawling on Google Maps to avoid abnormal behavior detection. While future development will be adjusted according to these findings, other tasks to be fulfilled in the future are listed as well, which will guide us to advance in the path of developing the project.
References


