Final Year Project Final Report

SpaceKey: Exploring Patterns in Spatial Databases

Author: Wang Jikun
Supervisor: Dr. Reynold C. K. Cheng

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

With the significant market size of the 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 halfway 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.

Acknowledgment

I would like to express my appreciation to all the people who have provided support 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 Contents

Abstract ........................................................................................................................................ 1

Table of Contents ......................................................................................................................... 2

1. Project Introduction .................................................................................................................. 6

1.1 Project Background ................................................................................................................. 6

1.2 Project Objectives and Requirements ..................................................................................... 6

1.2.1 Supporting APIs .................................................................................................................. 7

1.2.2 Dataset ............................................................................................................................... 7

1.2.3 A standalone demo housing application .............................................................................. 8

1.2.4 A project website ................................................................................................................ 9

1.3 Contribution ........................................................................................................................... 9

1.4 Report Structure .................................................................................................................... 9

2. Project Background and Literal Review .................................................................................... 11

2.1 The Property Searching Application Market .......................................................................... 11

2.2 Spatial Pattern Matching ....................................................................................................... 12

2.2 Other SGK queries .................................................................................................................. 13

2.3 (Appendix) Interest Survey ................................................................................................ 14

4. Project Methodology ............................................................................................................... 17

4.1 Data Collection ....................................................................................................................... 17

4.1.1 PoI Data ............................................................................................................................. 17

4.1.2 Property Data .................................................................................................................... 17

4.1.3 Data Storage ..................................................................................................................... 18

4.2 The APIs and Algorithms ...................................................................................................... 18

4.2.1 The API Server ................................................................................................................ 18

4.2.2 The Problem of Unwanted Objects .................................................................................. 19
4.2.3 The Problem of Realistic Distance ......................................................... 20
4.2.4 Directional Relations ........................................................................ 21
4.2.5 Custom Objects .................................................................................. 21
4.2.6 Additional Attributes .......................................................................... 22

4.3 Development structure and specs ........................................................... 22

5. Outcome and Results ................................................................................ 24

5.1 Data collection ....................................................................................... 24
  5.1.1 Crawler Implementation .................................................................... 24
  5.1.2 Difficulty ........................................................................................... 24
  5.1.3 Data Volume ....................................................................................... 25

5.2 The APIs and Algorithms ......................................................................... 26
  5.2.1 The API Server .................................................................................. 26
  5.2.2 The Problem of Unwanted Object ....................................................... 28
  5.2.3 Directional Relations .......................................................................... 29
  5.2.4 The Problem of Realistic Distance ...................................................... 30
  5.2.5 Custom Objects .................................................................................. 32

5.3 The Web Application ............................................................................... 32
  5.3.1 Front Page .......................................................................................... 32
  5.3.2 Search Page ....................................................................................... 33
  5.3.3 SPM Filter ........................................................................................ 35
  5.3.4 Custom Object .................................................................................. 37
  5.3.5 Routing Display ................................................................................ 38

6. Conclusion and Future Works .................................................................... 39

References ...................................................................................................... 40
List of Figures

Figure 1 illustrating the SPM query ................................................................. 12
Figure 2 survey result 1 .................................................................................. 14
Figure 3 survey result 2 .................................................................................. 15
Figure 4 survey result 3 .................................................................................. 15
Figure 5 survey result 4 .................................................................................. 16
Figure 6 the API server structure ................................................................. 19
Figure 7 development structure flow chart ................................................... 22
Figure 8 difficulty in the crawling process ................................................... 24
Figure 9 the request format of SPM ............................................................... 27
Figure 10 the comparison of efficiency of 3 approaches ............................... 29
Figure 11 illustrating directional relations .................................................... 30
Figure 12 front page ....................................................................................... 33
Figure 13 search page overview ................................................................. 33
Figure 14 region selection ............................................................................ 34
Figure 15 simple filters .................................................................................. 34
Figure 16 sorting ............................................................................................. 35
Figure 17 detailed information 2 ................................................................. 35
Figure 18 detailed information .................................................................... 35
Figure 19 SPM filter panel ........................................................................... 36
Figure 20 SPM filter panel w/ keywords filled in .......................................... 37
Figure 21 custom object panel ...................................................................... 37
Figure 22 routing display ............................................................................... 38
List of Tables

Table 1 contribution list...........................................................................................................................................9
Table 2 list of requests ..............................................................................................................................................27

Abbreviations

1. PoI: Points of Interest
2. API: Application Programming Interface
3. URL: Uniform Resource Locator
4. UI: User Interface
5. SGK query: Spatial Group Keyword query
6. $m$CK: $m$-Closest Keyword Query
7. minSK: Minimum Spatial Keyword Cover
8. CoSKQ: Collective Spatial Keyword Querying
9. SPM: Spatial Pattern Matching
10. OSM: OpenStreetMap
11. MTR: Mass Transit Railway
1. Project Introduction

1.1 Project Background

The demand for property has reached a new level with the significant market size of the 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 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.

1.2 Project Objectives and Requirements

The project’s outlook is to integrate our project as a module with existing housing applications to do advanced house searching/filtering with the help of SPM. In this module, users can filter available housing locations according to their interest in surrounding keyworded Points of

\(^1\) 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.
Interests (PoI), such as supermarkets, schools, etc. They can specify the distance and directional constraints between the desired property and keyworded/user-specified locations by drag-and-drop actions and clicks. At the same time, it will provide more information related to the locations 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/filtering works, which includes all the above functionalities.

1.2.1 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 the following basic functionalities:

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

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

1.2.2 Dataset

The next objective is to obtain a demo dataset for the demo application. 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
  o Realistic distance: travel time, 24-hour traffic, etc.

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

1.2.3 A standalone demo housing application

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

• **Pattern Specification**
  The demo application will feature an easy-to-use and advanced pattern input interface. It will allow users to input the spatial pattern by dragging icons with cursor on computer or by hand on handheld devices. They can also specify distance and direction requirements by clicking/touching. The distance constraints specification will be straight-forward, with options such as near, far, etc. It will also allow users to specify other requirements for the location, such as rush-hour traffic, etc. In addition, other functionalities and options will be provided, such as inputting custom objects, etc.

• **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 for the user, one at a time. The user can look at other results by clicking buttons. Additional information regarding the locations will also be displayed if needed, such as traffic information, reviews, ratings, etc.

• **Additional functionalities/queries**
  The demo application will also enable other types of spatial keyword queries for comparison purpose.
1.2.4 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 the project’s plan, interim report, and final report;
- Some other materials related to the project, such as the libraries used.

1.3 Contribution

The work on the project is distributed to the team members in terms of the structure of the project. The team has divided the project into three main parts: the collection of the data, the application website and the API server that handles data queries and SGK queries. The following table shows how the team members have contributed to the project.

<table>
<thead>
<tr>
<th>Team Member</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budiman</td>
<td>• The development of the web application (70%)</td>
</tr>
<tr>
<td>Yin Yue</td>
<td>• The collection/crawling of data (property and PoI) (100%)</td>
</tr>
<tr>
<td>Wang Jikun</td>
<td>• The development of the APIs (100%)</td>
</tr>
<tr>
<td></td>
<td>• The implementation of the algorithms (100%)</td>
</tr>
<tr>
<td></td>
<td>• The development of the web application (30%)</td>
</tr>
</tbody>
</table>

*Table 1 contribution list*

Please be noted that my individual report is based on my personal knowledge and due to the distribution of the work in the project is based on workflow, and some of the work is not interrelated, I might not be able to perfectly express the part that I did not participate in, i.e. the data collection part. The corresponding sections are 4.1 and 5.1.

1.4 Report Structure

The remaining parts of this report were arranged as the followings. First, we will discuss some background of the project, including the current situation of property searching application
market, as well as literal review of the research we have employed in our application, specifically Spatial Pattern Matching and other SGK algorithms. Then, we will discuss the approach we took for the development of the whole project in three aspects: data collection, algorithms as well as the website development. After that, the results and experiments we have obtained during the project will be discussed, which also includes some difficulties and challenges we encountered during this period. The products, including the APIs and the website, will be discussed in detail in this section as well. Finally, a conclusion will be drawn for this report, and what can be improved in the future will also be discussed in this section.
2. Project Background and Literal Review

In this section, the background of the project, as well as the literal review of previous works of this project, specifically Spatial Pattern Matching, will be discussed.

2.1 The Property Searching Application Market

The population in Hong Kong have been constantly rising. According to the statistics from the Hong Kong government, there is an increase of about 1.8M, from 5.5M in 1986 to 7.3M in 2016. Within this population’s growth, the number of expats is also increasing from 0 to 36k in 2016. Newborn people will need their own living space eventually. Likewise, expats need some living space to accommodate him/her and possibly with his/her family. Therefore, we can the reason why the demand for property has reached a new level with the significant market size of the 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. Taking one of the property-searching websites as an example, on this website, 28hse.com, the user can specify the property’s location. In addition, the user can add some filters regarding the house, such as its price, number of bedrooms, living area, and the buildings’ age. After investigating these properties searching applications, observations show that these applications only support simple searching functionalities, by searching for single elements and applying simple filters\(^2\). 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, hence is why we decide to utilize the technologies described in the following section to enhance these property-searching functionalities.

\(^2\) 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.
2.2 Spatial Pattern Matching

With location-based services prevailing in many real-life applications such as Google Maps and Flickr, spatial keyword search as a major problem model in this area has received plenty of research interest for the past few years. This problem model is defined by queries on spatial objects associated with textual information represented by sets of keywords. It combines the spatial and the text features of the map and aims at finding object(s) according to the distance and relevance of the keywords. Amongst various types of spatial keyword query, Spatial Group Keyword (SGK) query, which refers in particular to spatial keyword queries that returns a group of objects as its result, is a very important class that serves various real-life use.

Existing query types in this class (e.g., mCK) aim at finding a set of objects that are relevant to a set of input keywords, and whose locations are close to each other. However, the distance constraints that the user may want to specify could be more general. For example, when a user wishes to search for a house, she may expect a school nearby, which is close to her house but not too close (e.g., to avoid the noise and crowd caused by school). In this case, traditional SGK queries are not applicable. To tackle this type of problem, Spatial Pattern Matching (SPM) is developed.

Each query includes more detailed distance constraints in addition to a set of keywords, including the required lower-bound of the upper-bound of the distance between each pair of keywords. This problem model is the major model to apply in our application.

![Figure 1 illustrating the SPM query](image-url)
To be specific, such query can be illustrated with an example shown in Figure 1. Suppose the user wants to search for a house with a school nearby, but he does not want the school to be too close either because of the noise and traffic around the school. Therefore, he wants it to be farther than 0.3km and close than 1 km. In this case, we will draw a line between house and school with the distance constraint, as is shown in Figure 1(b). The same applies to the park and the station in the figure, and this figure represents the spatial pattern we are searching for on the map. In a spatial object set like such shown in Figure 1(a), the only suitable match is demonstrated in the figure linked by lines, with the appropriate distance between all its components in the spatial pattern.

2.2 Other SGK queries

As is shown in the objectives, for our APIs as open web service, we will also implement other SGK query types to enrich the diversity of our API set. The following types are to be implemented in our API set, see section 5.2.1 for query format.

**m-Closest Keyword Query (mCK):** Each query includes a set of keywords. The solution object set need to cover all the keywords in the queried keyword set, while minimizing the maximum pair-wise distance.

**Minimum Spatial Keyword Cover (minSK):** Each query includes a set of keywords. The solution object set needs to cover all the keywords in the queried keyword set, while minimizing the maximum pair-wise distance * (|objects| - 1) or the maximum pair-wise distance * C (|objects|, 2), to make the resulting set is as small as possible.

**Collective Spatial Keyword Querying (CoSKQ):** Each query includes a query location and a set of keywords. The solution object set need to cover the all keywords in the queried keyword set, while minimizing a specified cost function related to the query location and the objects’ location.
2.3 (Appendix) Interest Survey

In order to further understand and analyze users’ interests on the current housing applications as well as this newly proposed functionality, we went out and did an interest survey and received 82 responses.

About the samples of this survey, there are 56.1% female and 42.7% male. Since the survey is mostly done by the students the team members acknowledge, we can see all the samples are young people: 73.2% of the samples are the age of 17-22 and the rest are all age of 23-30. 85.4% of the samples are currently staying in Hong Kong or planning to stay in Hong Kong, which indicates that the samples are potential users of housing applications and websites.

When they are asked about the extent they find the current property searching applications and websites, we can see that the opinions diverge as is shown in figure 2: only 8.5% of the samples rated 5 and thinks that these property search applications are perfectly satisfactory to use; about half (46.3%) rated 4 and thinks they are great but they still have some small flaws, and the rest half of the samples rated only 1~3 out of 5. This result shows that the current property searching applications such as airbnb.com/28hse.com are still not perfect and there is a lot of room to improve.

![Figure 2 survey result 1](image)

Then, since our application focuses heavily on the location and surrounding points of interests of a property, we asked about what are the prime features that matter in the consideration when choosing a property in a property searching application and included location and
neighborhood as two of the choices. According to figure 3, we can see for these young potential users, among the 5 features of a house, overall one would highly value price, location and cleanliness. The neighborhood comes next with a lower number of ‘strongly agree’, but it has the highest number of ‘agree’. The decoration comes last with the lowest agree-disagree ratio. From this result, we can acknowledge that our new feature can have a great demand, since the location of the property and the neighborhood (surroundings) are considered very important features of a property.

![Figure 3 survey result 2](image)

Then, we asked about if they are interested in a feature that could allow them to specify the distance between the property and the places near it, which is essentially the core of our feature. According to figure 4, we can see that 79.3% of the samples rated their interest 4 or 5 out of 5, from which we can conclude that most people are positive about the feature.

![Figure 4 survey result 3](image)
Finally, we asked these samples about what types of points of interests is more important to them in the property’s surroundings. According to figure 5, we can easily see that these points of interest (PoI) varies in their importance. We can see that MTR stations, supermarket, convenience stores, restaurants and ATM are rated rather high among the list; Shopping mall, bank, bus stop and university are thought to be quite important as well. We can utilize this list to prioritize the list of keywords and the PoI data in our application.

Figure 5 survey result 4
4. Project Methodology

In this section, we will discuss the approach we took for the development of the whole project in three aspects: data collection, algorithms as well as the website development.

4.1 Data Collection

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

4.1.1 PoI Data

PoI data is collected from the Google Maps website by crawling. Python libraries such as Selenium are utilized to simulate interactions by controlling the browser through web drivers. This crawling process starts from initiating a search on Google Maps with a manually chosen keyword (e.g. School, Hospital) on the web driver. Then the plus code\(^3\) in the result is passed to another web driver for translation into coordinates using Google Maps. Finally, the result collected is formatted for storage in the database. Chrome web driver 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 a similar procedure.

4.1.2 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 an 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

\(^3\) Plus code is a combination of shortcode and locality which describe a unique location. For instance, 9CG+2X Noida.
property will also be stored along with the property specifications (e.g. price, area, region) for redirecting links and uniqueness validation.

4.1.3 Data Storage

Data formats are defined as database tables with a null value as the 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, 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 the 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.

4.2 The APIs and Algorithms

As is discussed in section 1.2, one of our objectives is to make a set of APIs that is separated from the web application that can operate on its own and serve as an open web service for developers to utilize. The following section will first discuss the approach we took for developing our API set. In addition, in this project, SPM is the major problem model of our application. In order to apply SPM to our project, we need to fit the original SPM theory to the real scenario. The following sections will also be discussing some ideas to make better use of the existing algorithms, as well as some challenges and respective approach we took when directly applying the SPM problem model to our project.

4.2.1 The API Server

The API is set to be developed in Java using Spring. It mainly contains 2 set of controllers that serve for different purposes as is structured in the figure below.

---

4 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.
- Data controller: Handles requests only related to the database, such as filtering and sorting the data according to certain specs.
- Algorithm controller: Handles requests that are related to algorithms, including:
  - Building index structures for (spatial) data in the database;
  - Spatial Pattern Matching (with variances)
  - Various other types of SGK for the purpose of comparison

Separating the controllers gives us the freedom to control the openness of our APIs. We want to set up the APIs in the cloud as an open API service, and as is shown in the figure, in this way we can expose the algorithm controller without exposing our data. Developers will be able to use the API to experiment and build future apps, while they won’t be able to access our data, while our website application will utilize both controllers in our API through keys.

### 4.2.2 The Problem of Unwanted Objects

Suppose we have a user really does not like airports to be around his/her property because it is too loud. As a result, to specify a spatial pattern for such a request, the user will add the keyword ‘airport’ with its distance constraints set to be between some very big number of infinity. In
this way, the properties returned will not have an airport nearby indeed, however, the efficiency is suffering.

The reason is that in the original SPM algorithm, when a keyword is specified in the pattern, the returned result must contain a PoI with that keyword regardless of the magnitude of the distance constraints [2]. Assume in the case above we set the distance constraint to be [100km, ∞], the algorithm will search for an airport instance that is farther than 100km away. By matching an instance that is so far away, the range that SPM needs to operate on increases enormously and as a result the efficiency will suffer. This is unnecessary since the user does not care about an actual instance of ‘airport’ but he just wants to be away from it.

Therefore, we want to propose some solutions to solve this problem and improve our performance in this case. We will first separate the ‘unwanted’ option as an individual option for distance constraints in our application. Experiments will be conducted to test and compare the performance of different approaches on a big dataset and our application will employ the optimal one according to the test results.

4.2.3 The Problem of Realistic Distance

In the original SPM algorithms, the distance between locations is measured by Euclidean distance, which is not realistic. This might make it not as useful since locations that are geometrically closer to each other might take more time to travel between. In the case of two locations that both sit very close to MTR stations, it takes far less time to travel between than two locations that sits alongside a mountain pass, even if they are relatively close to each other. Therefore, we want to explore the possibilities to employ more realistic distance measurement in our application, such as travel time (using car/public transport) and route distance.

In order to obtain routing information between locations, we need to use existing map services and APIs, such as Google Maps API, OpenStreetMap (OSM), etc. We will test on these map service options and see whether they are practicable in our application. Also, the implementation of using this distance needs to be discussed and tested too, such as whether to request the routing information on the go during the process of SPM or localize all the routing data first and store them in the database, etc.
4.2.4 Directional Relations

In the original SPM algorithms, directional relations are not considered. However, in a real-world scenario, directional relations might be useful when choosing a property: Suppose we have a user that wants a temple to the north of his/her property for the sake of fortune (风水). This seems to happen quite often, especially in the case of Hong Kong, therefore we decide to achieve this functionality in our application.

This functionality can be achieved through different implementations, such as by adding a layer of comparison in the core process ‘MStarJoin’, or by applying a filter on the result set to validate directional relations, etc. We will choose one implementation in our application according to their performance and implementation difficulty, which will be discussed in section 5.2.4.

4.2.5 Custom Objects

Suppose we have a user that wants to find a property to have strong ties to some specific locations, for example, he/she wants to have a property that is close his workplace as well as his friends’ house. In this case, SPM itself won’t do the job, therefore, as another additional functionality, we want to enable users with the ability to specify custom objects in the spatial pattern. This means that the user can add temporary objects to the PoI database with its unique keyword, which enables the user with the ability to specify the distance/directional constraints between his/her desired property and the custom object.

In our project, the user will be able to specify a custom object by clicking on the map or specify latitude and longitude, as well as assigning a name to the custom object. Then we can add the object temporarily to the PoI database by assigning a unique identifier as the keyword of this object, so that any result returned must include the specific location. Further discussion will focus on how the custom objects are added to the PoI database, specifically the index structures for spatial information, to optimize the efficiency.

In addition, more useful information could be displayed related to the custom objects, since they are rather special and has a higher priority compared to PoIs. For example, in the case of workplace, detailed routing information, such as travelling time for different transportation,
24-hour traffic information (especially rush hours), will be quite useful for the user. This information could be obtained from various existing map APIs and we want to explore the possibilities therein.

### 4.2.6 Additional Attributes

Similar to 4.2.5, 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.

### 4.3 Development structure and specs

![Development structure flow chart](image)

**Figure 7 development structure flow chart**

- **API set:**
  The supporting API set is developed in Spring Framework 5.0 using Java. It will be later hosted either on a cloud service such as AWS or on our own dedicated machine as a local server.

- **Database:**
  The database of properties and PoIs is developed and maintained in MySQL in separate with the rest of the components. It is suited to be hosted either on a cloud service such as AWS or on a dedicated server.
• **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 is used. For the map, google-maps-react is used as the primary tool.
5. Outcome and Results

In this section, we will discuss our outcome and results we have obtained in our project. Sections are similarly divided into data collection, API and algorithms and the web application. In the data section we will discuss what we implemented to obtain the data, the challenges we overcame and the resulting dataset we have in the end. In the API and algorithms section we will discuss the results we have implementing the ideas described in section 4.2 one by one. In the web application section, we will introduce the product website in detail.

5.1 Data collection

This section will cover the experiments and results we have done and achieved in the process of data collection, as well as the difficulty we encountered.

5.1.1 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 an accidental shutdown. Crawlers are deployed on AWS EC2 server instances to achieve automation and concurrency.

5.1.2 Difficulty

![Source: https://www.google.com/maps/@22.2910051,114.1424312,12.97z](https://www.google.com/maps/@22.2910051,114.1424312,12.97z)

*Figure 8: difficulty in the crawling process*

---

5 AWS EC2 is Amazon Elastic Compute Cloud, a web service that provides secure, resizable compute capacity in the cloud.
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 a 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 the crawler, therefore the feasibility requires further investigation.

During the process of trying to scrape and extract results from map websites, we found it especially difficult to extract results from the URL response normally given by these websites. The main problem is that the AJAX response received from a Google Maps API query is not well-formatted. Therefore, understanding the format and parsing an additional necessary step to the process, which is extremely time-consuming and difficult. After studying about the response, we found out a feasible way to avoid this step, which is to rely on the HTML result generated from the response using the selenium library. This library supports locating element and simulating user-interaction with the browser, therefore enabling us to skip the process of parsing the AJAX response into HTML object.

5.1.3 Data Volume

According to the database record count, the collected data volume is 39,909 entries of PoIs from Google Maps. Additionally, there are about 17,000 entries of properties for sale and about 24,000 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.
5.2 The APIs and Algorithms

In this section, we will discuss the progress of the application of SPM, including the implementation of the APIs, as well as how we have attempted to address the ideas technically previously mentioned in section 4.

5.2.1 The API Server

Following the approach mentioned in section 4.2.1, we have implemented the API set, which is currently ready to deploy in any server environment. The following table shows the requests it can handle currently.

<table>
<thead>
<tr>
<th>Controller</th>
<th>URL</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Controller</td>
<td>/data/property/get?type=X&amp;region=Y</td>
<td>Request for properties of type X (sell/rent) that are located at region Y</td>
</tr>
<tr>
<td></td>
<td>/data/property/all</td>
<td>Request for all the properties</td>
</tr>
<tr>
<td></td>
<td>/data/poi/get?keyword=X&amp;region=Y</td>
<td>Request for points of interest of keyword X that are located at region Y</td>
</tr>
<tr>
<td></td>
<td>/data/poi/all</td>
<td>Request for all the points of interests</td>
</tr>
<tr>
<td>Algorithm Controller</td>
<td>/alg/spm_simple</td>
<td>Run SPM algorithm on the current data set. Parameters are in the request body.</td>
</tr>
<tr>
<td></td>
<td>/alg/mck</td>
<td>Run mCK Exact algorithm on the current data set. Parameters are in the request body.</td>
</tr>
<tr>
<td></td>
<td>/alg/minsk</td>
<td>Run minSK Scale Lune algorithm on the current data set. Parameters are in the request body.</td>
</tr>
<tr>
<td></td>
<td>/alg/coskq?type=X</td>
<td>Run CoSKQ Exact algorithm with type X cost on the current data set. Parameters are in the request body.</td>
</tr>
<tr>
<td></td>
<td>/alg/spm_topk</td>
<td>Run SPM algorithm with results top k sorted on the current data set.</td>
</tr>
</tbody>
</table>
A request to run an algorithm is required to have an organized set of parameters, and the parameters are easily extensible when we add new functionalities to the API. Taking spm_simple as an example, there will be required/optional parameters as is shown in the figure:

A set of wanted objects represents the set of PoI one wants to have/avoid around his property. Each PoI has its keyword as the identifier of this PoI, lower/upper represents the distance constraints the user wants to have between his/her property and the PoI, ‘dir’ represents the direction the user wants it to be in. As we can see a set of custom objects is optional to the request, which is an added feature to the original SPM that we will discuss later. Whenever there are some additional features, we can easily extend the parameter set by modifying the request identifiers in the controllers.
5.2.2 The Problem of Unwanted Object

Upon this challenge mentioned in 4.2.2, we have defined the problem and proposed three potential approaches to deal with this problem. Consequently, experiments are conducted, and we choose an optimal approach with better efficiency in our application. 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 (+inf);

2. Use the SPM on P \ 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 \ 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 - 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 a 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 on the map. In this case, approach 1 will return nothing, which does not satisfy the user’s request.

Approach 2 and 3 eliminate this problem because instead of directly using the SPM algorithm with P, they use P \ 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 - approach 2 is filtering results by looking at unwanted keywords, while approach 3 does it by performing SPM on the result set with every additional unwanted keyword. Therefore, the meaning behind these two approaches is equivalent. In addition, if all the unwanted keywords exist on the map, approach 1, 2, and 3 have the same problem to solve, therefore they are equivalent in this case.
In terms of efficiency, we have managed to test all three approaches in the UK PoI data with keywords. The spatial database is larger-sized than the HK PoI database we collected, which contains 180,000 PoI, each of them contains a set of different keywords. It is more suitable for the purpose of testing and thus we chose to use it instead of our own database. In our test, we perform 1000 such queries with unwanted keywords randomly selected from the top 50 popular keywords in the database, and for each approach the set of queries is the same. During the queries we record the average time spent on each query for the three approaches. As is shown in the figure, the second approach has a slight edge upon the first one, and both are far better than the third approach. After analyzing the querying process, we found that 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, according to the experiments and analysis discussed above, we opt for the second approach in our project for its better efficiency and applicability for all situations. This issue is left now for future discussion and whenever a better approach rises, we will adjust our approach in our application accordingly.

5.2.3 Directional Relations

As discussed earlier in 4.2.3, we want to develop a filtering functionality in the SPM filter that gives users the ability to choose which direction they want their PoI to be in. This functionality can be achieved by adding another layer of comparison in the process of SPM on the level of
index structures to validate if a certain pair of points satisfy the directional relations. This process does not vary the efficiency of the querying process.

![Figure 11 illustrating directional relations](image)

As a result, we have successfully implemented the idea in the SPM functionality for our application. As is shown in the figure above, which is from the SPM specification display in our application, we categorize all the objects in the fan-shaped area (-45°, +45°) as they are in the same direction. Thus, we only employ north, south, east and west as options, and when the distance constraint differs, the ‘fan’ changes its shape and position in the display.

**5.2.4 The Problem of Realistic Distance**

The team has attempted to solve the problem of realistic distance discussed in section 4.2.4. However, several limitations have denied the possibility of doing so for our current application.

We first investigated the SPM algorithms. The original SPM algorithm is using Euclidean distance as the standard of evaluating distances. It only requires the latitude and longitude of points and it takes no time to calculate. However, if we want to switch it to realistic distance, we need to request the possible routings from data sources online, such a Google Maps API, OpenStreetMap (OSM). Not only it takes time to request such routings, after investigating these APIs, none of them provide free-of-charge routing service to developers/users. During the investigation of the process of solving SPM, we can see there will be up to 100,000 evaluations
on distance in each query, and it is certainly unrealistic to request them on the go during the process of solving SPM, since it will take an extremely long time. Localization of such routing data is needed.

Then we proceeded to request these routing data and store them in our local database. We calculated the time needed to request routing data between a pair of property and PoI – it takes about a day to collect 10,000 entries. Suppose we have only 1,000 property data and 1,000 PoI data, there need to be 1,000,000 entries of routing information and it will take 100 days to collect these data if our program runs non-stop, therefore collecting all the routing info becomes unattainable. In addition, such amount of data localization will have both technical issues and licensing issues since it is not allowed and can be detected by the data providers. Therefore, we gave up this thought of directly using realistic data as our standard of distance constraints.

Due to the routing data source limitations mentioned above, we proposed and implemented alternative ways to employ realistic distance. Same as before, Euclidean distance will be still used in the process of solving SPM. Since realistic distance is always larger than Euclidean distance, we make the distance intervals specified by the user slightly looser. By running SPM on these looser constraints, there is a high chance that what the user wants for the realistic distance he/she specifies is still in the results we return. Then, the routing queries on a map API will be used to get the realistic distance on these results, which is a much smaller number of queries compared to the direct implementation we have shown above. Finally, we filter the results by comparing their realistic distance to the constraints the user initially specified. In this way, realistic distance evaluation becomes attainable. However, though this is what we have implemented, it is hard to evaluate the accuracy and effectiveness of the results as compared to directly using the Euclidean distance as the evaluation method. This is left to be discussed in future investigations.

In addition, such routing information is also used to enrich the functionalities of our application, as we give users the option to display the routing information between their chosen property and PoIs. See section 5.3.5.
5.2.5 Custom Objects

As discussed in 4.2.5, we want to enable users with the ability to add user-defined PoI temporarily in order to facilitate their search for desired properties. We have implemented this function in the API. The application now allows users to add multiple custom objects with user-defined tags through a comprehensive user interface, by dragging and clicking on the map (To see the user interface and how to use this functionality, see 5.3.4). When the user specifies a spatial pattern with custom object(s), the user-defined tag will be used to match this object. First, in order to give the uniqueness this custom object needs during the search (in case it matches other identically-named PoIs), we encode the tag as its identifier during the process of SPM. If the user does not want to define a name for the custom object, we generate a random string that is unique among all the keywords in the database. Then in order to add this new object into our index structure, we need to partially reconstruct the index structures. Note that at first complete reconstruction of the index structures were applied, but since it is called every time the query is requested, it takes a lot of time to do so, and therefore partial reconstruction is used, and it is much more efficient than reconstructing the whole structure. After successfully updating the index structures, this spatial pattern with custom objects can be treated as a normal spatial pattern.

More functionalities are developed based on the potential demand for custom objects, such as routing request and display, see 5.3.5 for more details.

5.3 The Web Application

This section will discuss the functionality and design of our website application.

5.3.1 Front Page

The front page is a simplistic GitHub-style webpage that serves as the project website specified in the first deliverables and is continuously updated during the progress of the project. In the center it carries the project title SpaceKey, as well as a button redirecting the user to the search page. This page also carries a short introduction of the project, as well as links to documentation such as API usage and project reports. Promotions are expected to be placed on this page in the future.
5.3.2 Search Page
This page on the main page of our application. It contains several components:

- Region Selecting Bar: In which the user can select the region he/she wants to search the property. All the regions are in Hong Kong with the exception of Japan. This list is auto-generated according to the database, as is shown in the figure above. After selecting the region, all the properties in that region will appear on the map. It is set as a single element since it is the first thing to do to initialize search. Similar styling can be found in Airbnb, VRBO, etc.

- Filter Bar: In which the user could specify their desired property. It contains several simple filters, such as the number of bedrooms, saleable area, gross area and price, and one can adjust them with ease, as is shown in figure 15. It also contains a button for SPM, clicking which will turn up the panel to specify SPM as well as custom objects. It will turn red if it is activated, otherwise in grey. On the right it has a dropdown list for choosing the sorting methods, as is shown in figure 15.
• The Map: In which the results, including the properties and specified PoIs are displayed.

• Info Display: In which the detailed information of the property, as well as a picture of high quality are displayed, as is shown in figure 18 below. A link at the bottom will redirect the user to the original webpage on 28hse.com/Airbnb that we got our data from. Alternatively, when a property is clicked on the map, detailed information will also be displayed in a modal.

• Title Head & Bottom Info Bar

5.3.3 SPM Filter

By clicking the SPM button in Filter Bar, the panel for specifying parameters in SPM will pop up as shown in figure 19 below. In order to simplify the process of specifying distance constraints, we only allow users to specify 3 distance intervals, which stands for close, medium
and far respectively. They are by default 0-500m, 500-1000m, 1000m-1500m, but the users could modify these three intervals by dragging the circles along the axis. They are required to be consecutive intervals and it is not allowed to drag across one another. In the input bar below the user can search and add existing keywords to specify the pattern.

![Figure 19 SPM filter panel](image)

After the user types in and selects some keywords, it will show up below the input bar, as is shown in figure 20 below. Then the user can select the parameters related to that keywords. For directions, the user can choose north, west, east, west or any; For distance, the user can choose close, medium and far as they specified above, as well as any and unwanted. Unwanted means that the user does not want the keyword they specified to be any close to their property, which is an additional functionality discussed in section 4.2.2. On the right is a comprehensive pattern display, separating different distances and directions with lines. As the user add more keywords to the pattern, their respective area will be highlighted, and the keyword will be attached to the area in the display as well. When the specification of the pattern is done, the user can click the ‘Apply’ button to apply this SPM filter, then this panel will close and show the results on the map, including the properties and the PoIs that are related.
5.3.4 Custom Object

This functionality can be accessed inside the panel for SPM filter. First, the user needs to choose a region in the dropdown list and drag the map to their desired location. The user can type the name of his/her custom object and click the ‘Add’ button. Then the custom object will
appear at the center of the map. The object is draggable, so the user could finalize the position of their custom object. After adding all the custom objects that the user needs, he/she can switch back to the PoI panel and see all of their custom objects’ keyword have been automatically added into the pattern, waiting for the user to specify their distance and directional constraints.

5.3.5 Routing Display

![Routing Display](image)

Figure 22 routing display

We have enabled users to evaluate the realistic distance between properties and PoIs with this functionality of routing display. To use this functionality, when the results are displayed on the maps, the user needs to click on the property they want to route and click the ‘Select’ button below the detailed information of this property. After clicking it, the button will turn red and all the other properties will be hidden on the map, leaving only this property and the PoIs. Then the user needs to click on the PoI that is the destination. Then the routing information will be displayed on the map as a polyline. The user can start over by clicking on the Select button again.
6. Conclusion and Future Works

This 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. Throughout the development of the whole project, the team has encountered a lot of difficulties and limitations, but at the same time achieving the goals we set in each phase. For the data collection, we have collected and integrated a certain amount of property data from 28hse.com and Airbnb, as well as PoI data from Google Maps using different techniques; for the API and algorithms, we have used ways to optimize the SPM queries for it to be used in our application, and we have also added several additional useful functionalities; For the website, the team has fully designed and implemented the product, which is great for demo and promotion purpose. In the future, we hope that more features could be added to our website, and the application of SPM could be further improved. Also, we hope to further promote our ideas in this project to more people and hopefully it will be useful to some future development.
References


