SI^{2}P: A Restaurant Recommendation System Using Preference Queries over Incomplete Information

Xiaoye Miao[#] Yunjun Gao[#,†] Gang Chen[#,†] Huiyong Cui[#] Chong Guo[#] Weida Pan[#]

[#]College of Computer Science, Zhejiang University, Hangzhou, China

[†]The Key Lab of Big Data Intelligent Computing of Zhejiang Province, Zhejiang University, Hangzhou, China

{miaoxy, gaoyj, cg, cuihy}@zju.edu.cn {armourcy, iykoncoc001}@gmail.com

ABSTRACT

The incomplete data is universal in many real-life applications due to data integration, the limitation of devices, etc. In this demonstration, we present SI^{2}P, a restaurant recommendation System with Preference queries on Incomplete Information. SI^{2}P is capable of friendly recommending desirable restaurants based on preference queries that take the incomplete ratings information into consideration. It adopts the browser-server model, and incorporates three functionality modules including friendly and convenient *query submission*, flexible and useful *result explanation*, timely and incremental *dataset interaction*. SI^{2}P provides the server side based on an extended PostgreSQL database that integrates two types of preference queries, namely, skyline and top-k dominating queries over incomplete data. It also offers the browser-based interface for the users to interact with the system. Using a real restaurant dataset from TripAdvisor, we demonstrate SI^{2}P can recommend and explore the restaurants in a friendly way.

1. INTRODUCTION

The last decade has witnessed an increasing interest in preference queries. Motivations for such concern are manifold. First, it has appeared to be desirable to offer more flexible queries that meet the users' minds. Second, preference queries usually provide a basis for rank-ordering the objects retrieved, which is especially valuable in case of the large set of the objects satisfying a query. Consequently, several systems supporting preference queries have been built such as Preference SQL [5] and FlexPref [6]. Almost all the existing preference query systems rely on one assumption, i.e., data is *complete*. In other words, all dimensions are available for every data object.

However, it is obvious that, the incomplete data is universal in many real-life applications, as a result of data integration, accidental loss, incomplete responses in survey, the limitation of devices, user error, system failure, etc. Table 1 gives a real dataset which contains 12 restaurants from New York city crawled from TripAdvisor[1]. In this table, each restaurant is represented by its ratings information from customers including the price level, the number of ratings, and the ratio of positive ratings. It is highly likely that the ratings (e.g., price levels) of some restaurants are *incomplete*, if there is no customer scoring the price levels of the restaurants. For example, the restaurant "*The Lobster Place*" misses its attribute value on *Average price level*. It is worth mentioning that, if data entries with incomplete values are ignored, it makes some analytic queries fail to accurately describe how an organization is performing [1]. As a result, the management of incomplete data has triggered lots of efforts from database community, since the incomplete data incurs bigger challenges, e.g., non-transitive dominance relationship [3, 4, 7].

Table 1: A Real Dataset with Missing Data

Name	Average price level	Number of ratings	Ratio of positive ratings (%)
The Lobster Place	*missing*	1001	93.81
Colicchio & Sons	*missing*	1367	91.22
Club A Steakhouse	40	2176	91.31
Keens Steakhouse	40	3474	90.50
Taim	*missing*	175	94.29
Dinosaur Bar-B-Que	20	688	91.72
Num Pang Sandwich Shop	*missing*	171	94.15
Market Table	30	295	92.54
Basso56	30	1608	89.99
The Little Owl	30	793	90.16
Gramercy Tavern	40	2201	90.23
Mighty Quinn's	*missing*	298	90.94

In this demonstration, we build the SI^{2}P system, which employs the preference queries based on the incomplete rating information for restaurant recommendation. The supported queries of SI^{2}P are categorized into four types: (i) skyline queries on incomplete data, where users request a set of the objects that are not dominated by any other object; (ii) top-k dominating (TKD) queries on incomplete data, where users request a set of the top-k objects with the highest scores; (iii) range queries, where users request a set of the objects in a certain area; and (iv) basic operators in relational databases, such as selection, projection, join, update, etc. To the best of our knowledge, this is the first implementation for skyline and TKD queries on incomplete data into PostgreSQL, complementing the traditional skyline query implementation on PostgreSQL [2]. Our key contributions in the demonstration are summarized as follows.

- We develop a restaurant recommendation system based on incomplete data, namely, SI^{2}P, adopting the browser-server model. The server side is built based on the open source PostgreSQL database by integrating preference queries including skyline and TKD queries over incomplete data.

[1]Available at http://www.tripadvisor.com/.

Proceedings of the VLDB Endowment, Vol. 9, No. 13

- We improve SI^{2}P to support three interaction functionality modules including friendly and convenient *query submission*, flexible and useful *result explanation*, and timely and incremental *dataset interaction*.
- Using a real restaurant data set from TripAdvisor, SI^{2}P is demonstrated to recommend the representative restaurants in a friendly way.

The rest of the demonstration proposal is organized as follows. Section 2 defines the queries supported. Then, we introduce SI^{2}P system in Section 3. Section 4 offers the demonstration details. Finally, Section 5 concludes the demonstration.

2. DEFINITIONS OF QUERIES

The skyline query and the top-k dominating query on incomplete data are stated in Definition 2 and Definition 3, respectively. Both queries are based on the dominance relationship over incomplete data in Definition 1, where the symbol $o.[i]$ is used to represent the i-th dimensional value of an object o.

DEFINITION 1. ***(dominance relationship on incomplete data [4])***. *Given two incomplete objects o and o', o dominates o', denoted as $o \prec o'$, if the following two conditions hold: (i) for every dimension i, either $o.[i]$ is no worse than $o'.[i]$ or at least one of them is missing; and (ii) there is at least one dimension j, in which both $o.[j]$ and $o'.[j]$ are observed and $o.[j]$ is better than $o'.[j]$.*

DEFINITION 2. ***(skyline query on incomplete data [4])***. *Given an incomplete dataset S, a skyline query over S finds the set $S_G \subseteq S$ of the objects that are not dominated by any other object in S, i.e., $S_G \subseteq S$ satisfies that, for $\forall o \in S_G$, there is no object $o_1 \in S$ such that $o_1 \prec o$, and for $\forall o' \in (S - S_G)$, there is at least one object $o_2 \in S$ such that $o_2 \prec o'$.*

DEFINITION 3. ***(top-k dominating query on incomplete data [7])***. *Given an incomplete dataset S, a top-k dominating (TKD) query over S retrieves the set $S_G \subseteq S$ of k objects with the highest scores, i.e., $S_G \subseteq S$, $|S_G| = k$, and $\forall o \in S_G$, $\forall o' \in (S - S_G)$, $\mathsf{score}(o) \geq \mathsf{score}(o')$, where for $o \in S$, $\mathsf{score}(o) = |\{o' \in S - \{o\} | o \prec o'\}|$.*

Take the dataset in Table 1 as an example, the *smaller* the *Average price level* and the *larger* the *Number of ratings* and *Ratio of positive ratings*, the better the dimensional value. Hence, restaurant "*The Lobster Place*" dominates restaurant "*Dinosaur Bar-B-Que*" since "*The Lobster Place*" has larger values in attributes *Number of ratings* and *Ratio of positive ratings*, but with no price information. Restaurant "*Club A Steakhouse*" is a skyline object as it is not dominated by any other restaurant. In addition, the top-1 dominating query returns the restaurant "*The Lobster Place*" with the highest score 4, since it dominates four restaurants including "*Dinosaur Bar-B-Que*", "*Market Table*", "*The Little Owl*", and "*Mighty Quinn's*". Intuitively, given a set of restaurants satisfying some constraints (e.g., the location requirement), a skyline/TKD query could retrieve the most representative restaurants.

3. THE SI^{2}P PROTOTYPE

In this section, we cover the framework of SI^{2}P and then the browser and server sides.

3.1 The SI^{2}P Architecture

SI^{2}P adopts the browser-server model and is implemented with Javascript and Python, and uses a PostgreSQL database. The architecture is shown in Figure 1, which details the interaction modules supported by SI^{2}P including *query submission*, *result explanation*, and *dataset interaction*. The user (requester) submits a query

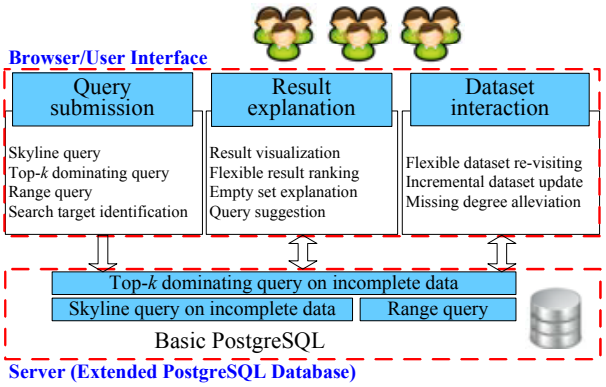

Figure 1: Architecture of SI^{2}P

through the *user interface* (via browser and Google Maps), and the query is then sent to the server for processing. After the query is processed, the results are sent back to the user who issues the query, and are displayed on Google Maps in the user's browser. In addition, SI^{2}P provides two open components that extend basic PostgreSQL to support skyline and TKD queries over incomplete data (available at https://github.com/SI2P/ SI2P-pg). Users can employ these components to develop their application systems.

3.2 Browser Side

The browser side in computers offers the interfaces to users for generating queries, viewing the returned objects, writing reviews on the restaurants. This browser shows a map, and provides interactions with the map using the Google Maps API.

When generating a query, the users can specify an interested region and some constraints (e.g., price level, cuisines) by drawing a rectangle in map and clicking/selecting keywords to form a query. SI^{2}P supports *search target identification*, which can identify users' common keywords, and help to save many typing efforts significantly. Note that, among the three queries including skyline query, TKD query (with k = 20 by default), and range query, the users need to specify which query they would like to select. The users can also delivery the price and cuisines requirements as the constraints for skyline query, TKD query, and range query. The specified query is then sent to the server for processing.

After the query is processed by the server, the objects retrieved are returned. How to visualize the query results properly is a crucial part in enhancing search experience. In particular, how to support *zoom in/out* for users to view results at various granularity of details is a challenge. SI^{2}P supports *visualized search object*, a mechanism for users to manipulate and interact with the result objects, to further explore all related information connected with that result object, and finally to find their desired information flexibly. In order to maximize the usefulness of visualization, in addition to displaying the result objects in the result list, SI^{2}P combines the query results with the Google Maps API, which can visualize the query results within a global context of the location (coupling with the user's targeted location). Furthermore, SI^{2}P supports *flexible result ranking*, a mechanism for users to sort the query results based on their own preferences.

Explaining the empty result set or the mismatch result can help users understand how the problem occurred. The browser side provides the simple *explanation* for the empty result set, e.g., there is no restaurant within a specified region (and/or satisfying the keyword constraint). What is more, SI^{2}P supports the *suggestion* for the empty result set, which recommends the most popular restau-

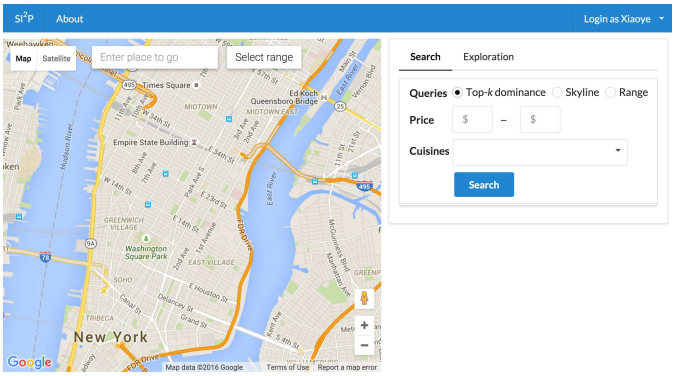

Figure 2: The browser interface of SI²P

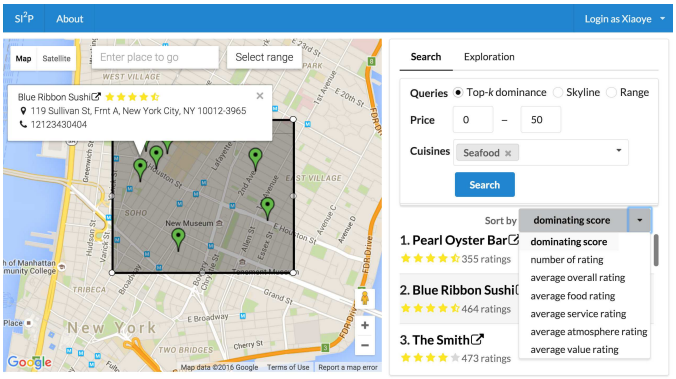

Figure 3: Illustration of submitting a TKD query

rants in the city the selected region located. This suggestion is also very useful when the returned result set is dissatisfying.

Apart from complementing the traditional lookup task mentioned above, another important type of information need from the users is to learn from the data or the result for a better understanding of the data they are playing with. To this end, SI²P offers the range query with some constraints for *dataset re-visiting*. It is important to note that, the users can identify the existing restaurants within a specified area using range query. Meanwhile, the detailed information of the restaurants can also be found via clicking the restaurants, as mentioned previously.

SI²P allows the users to write reviews on the restaurants, and score the restaurants such that the information of restaurants is updated *incrementally*. Specifically, the access control via login is implemented in SI²P, which identifies the users, and allows the *real* users to rate and post comments for the restaurants. As a result, once reviewing a restaurant, the information of the restaurant is updated. It is worth pointing out that, this mechanism can also *alleviate* the missing degree of the dataset. In addition, SI²P supports the users to *like* the restaurants, which could be used to identify intuitively the popularity of the restaurants for the users.

3.3 Server Side

The web server in SI²P is built by using Flask[2]. The dataset is stored in a PostgreSQL database (version 9.4). In addition to supporting the typical operators including selection, projection, deletion, and update, SI²P also integrates the skyline query and the top-*k* dominating (TKD) query over incomplete data into the PostgreSQL database. When a new query request is received by the server, it invokes the functions of PostgreSQL including the basic operations as well as the integrated skyline or TKD query.

We extend the PostgreSQL database by integrating the skyline query on incomplete data via IkSB algorithm [3] and the TKD query over incomplete data using UBB algorithm [7]. IkSB algorithm employs the bucket structure to partition the incomplete data objects into several buckets such that the objects in the same bucket have the same missing dimension(s). Then, it derives the *local* skyline objects for each bucket. Finally, the global skyline objects are returned via the selection from the local skyline objects. UBB algorithm utilizes the upper bound scores of the objects to determine the access order of objects to reduce the candidate set size. In particular, UBB integrates the ranking process into the object evaluation, and enables an early termination of TKD query processing before evaluating all the candidates. The presentation of the more details of IkSB and UBB algorithms can be found in [3, 7].

[2]Available at http://flask.pocoo.org/.

The IkSB and UBB algorithms are implemented in C and PL/PGSQL as *PostgreSQL extensions*, which can be loaded into PostgreSQL at runtime. The extensions are basically custom functions that define parameters and query processing. As a mild coupling external module, the extensions can be easily distributed and maintained. To be more specific, skyline and TKD queries on incomplete data are defined by functions `SKYLINE(dataset)` and `TKD(dataset, parameter_k)`, respectively. For the purpose of illustration, the function

```
SKYLINE(SELECT restaurant_name, price[min],
          numer_of_ratings[max], positive_ratio[max]
        FROM restaurant)
```

defines a skyline query on incomplete data which returns, from the table `restaurant`, the restaurants that are not dominated by any other restaurant, with the name of restaurant, the price, the number of ratings, as well as the ratio of positive ratings, where it is deemed that, for the preferences, the lower the price, the better. So is it for the more the ratings and the higher the positive ratio. These preferences are used in the dominance relationship judgement for the skyline query, as defined in Definition 1.

On the other hand, the function

```
TKD((SELECT restaurant_name, price[min],
          numer_of_ratings[max], positive_ratio[max]
     FROM restaurant), 12)
```

defines a TKD query on incomplete data which retrieves the top-12 restaurants from the table `restaurant`, with the name of restaurant, the price, the number of ratings, as well as the ratio of the positive ratings.

In addition, it is worth noting that, the PostgreSQL with the extensions can support *constrained* skyline/TKD query over incomplete data, i.e., the skyline/TKD query is conducted based on a constrained dataset which is shaped by users' preferences. For instance, the real dataset depicted in Table 1 is actually derived by invoking the following function,

```
TKD((SELECT restaurant_name, price[min],
          numer_of_ratings[max], positive_ratio[max]
     FROM restaurant
     WHERE restaurant_location is in New York), 12)
```

where the original `restaurant` table contains 61,146 restaurants within America crawled from TripAdvisor.

4. DEMONSTRATION

In our demonstration, we utilize a real-life restaurant data set crawled from TripAdvisor to give users the opportunity to interact

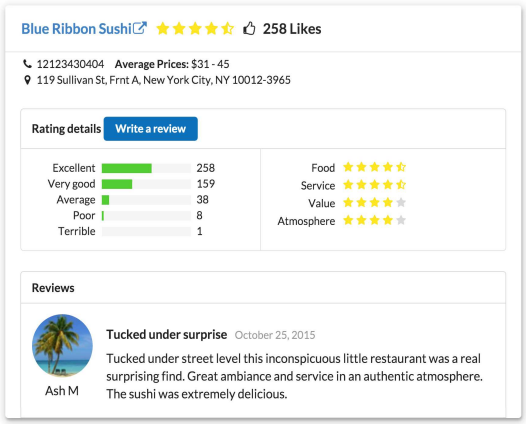

Figure 4: The detailed information of the restaurant "*Blue Ribbon Sushi*"

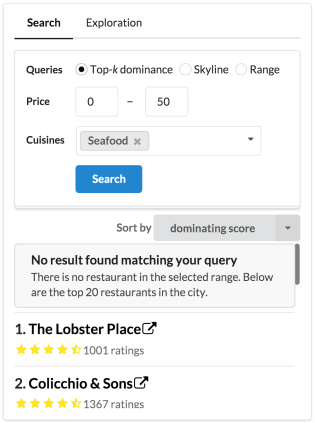

Figure 5: The explanation for an empty result set

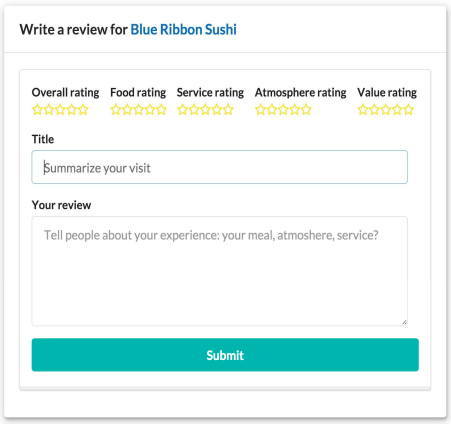

Figure 6: Illustration of writing a review

with SI^{2}P, and experience how the system can be used for retrieving the restaurants. The browser interfaces of SI^{2}P are shown in Figure 2. Users can use the system to find the interesting restaurants and review the restaurants. SI^{2}P is available at `http://115.28.42.70`. We prompt the audience to become data analysts and perform exploratory analysis.

Query submission scenario. The audience can specify the query from skyline query, TKD query, and range query via a single selection box. To specify an interested location, the audience can enter a coarse location and then select a fine-grained location by drawing a rectangle on Google Maps (the latitude and longitude of the location are obtained using the Google Maps API). The system also provides options for users to express the preferences. As exemplified in Figure 3, for the selected location from the map, the user chooses TKD query, and specifies the price interval as [0−50$] and the cuisines including *Seafood* as the constraints of the query. It is important to note that, skyline and range queries can be processed in the same manner, and hence, we omit the detailed presentation for space limitation.

Result explanation scenario. When the query results are returned to the browser of the users. The result objects are displayed in both the map and the result list. For example, the result of the above submitted TKD query includes "*Pearl Oyster Bar*", "*Blue Ribbon Sushi*", "*The Smith*", ⋯, as shown in the bottom right hand side of Figure 3, as well as within the shadow rectangle area of map in Figure 3. The participants could change the sort function of the result restaurants to view different recommendation results. Specifically, the users can choose the ranking function via a drop-down list, as shown in the bottom right hand side of Figure 3, where seven functions are provided including sorting by dominating score, by the number of ratings, by average overall rating, by average food rating, by average service rating, by average atmosphere rating, and by average price rating. Further, as exemplified in Figure 4, the participants are able to get the detailed information of the restaurant "*Blue Ribbon Sushi*", including the telephone, the address, the ratings, and the reviews from customers, via clicking this restaurant.

For some scenarios, the query result set turns to be empty. As exemplified in Figure 5, SI^{2}P explains the direct reason of the empty result set, i.e., there is no restaurant located into the selected range. Alternatively, SI^{2}P provides the top-20 restaurants derived from the TKD query in the city where the selected range is located.

Dataset interaction scenario. If the participants feel dissatisfying for the query results, they are allowed to view the popular restaurants in the city the selected region located. Moreover, they can overview the dataset (i.e., the restaurants) via range query under any constraint, and learn from the restaurant information.

In addition, if the participants login SI^{2}P system, they can score the restaurants, and write reviews on the restaurants, as illustrated in Figure 6. Furthermore, SI^{2}P allows the users to give a thumbs up to their favorite restaurants by clicking the "thumb" symbol, as shown in the top right hand side of Figure 4. Hence, the dataset changes with time going, and the rating information becomes more and more complete.

5. CONCLUSION

In this demonstration, we present our SI^{2}P system for restaurant recommendation using preference queries on incomplete data. SI^{2}P adopts the browser-server model. In the server side, SI^{2}P is based on an extended PostgreSQL database, which supports skyline and TKD queries on incomplete data. In the browser side, in addition to supporting convenient and flexible query submission and query result presentation, SI^{2}P has the functionalities including useful result explanation and dataset interaction.

Acknowledgements. Yunjun Gao is the corresponding author of this work. This work was supported in part by 973 Program 2015CB352502, NSFC Grants 61522208, 61472348, and 61379033, and the Fundamental Research Funds for the Central Universities. We are grateful to Jingda Chen, Ruijia Mao, and Danyang Song for the contributions on SI^{2}P system.

6. REFERENCES

[1] C. Christodoulakis, C. Faloutsos, and R. J. Miller. Voidwiz: Resolving incompleteness using network effects. In *ICDE*, pages 1230–1233, 2014.

[2] H. Eder and F. Wei. Evaluation of skyline algorithms in PostgreSQL. In *IDEAS*, pages 334–337, 2009.

[3] Y. Gao, X. Miao, H. Cui, G. Chen, and Q. Li. Processing *k*-skyband, constrained skyline, and group-by skyline queries on incomplete data. *Expert Systems with Applications*, 41(10):4959–4974, 2014.

[4] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. Skyline query processing for incomplete data. In *ICDE*, pages 556–565, 2008.

[5] W. Kießling and G. Köstler. Preference SQL: Design, implementation, experiences. In *VLDB*, pages 990–1001, 2002.

[6] J. J. Levandoski, A. Eldawy, M. F. Mokbel, and M. E. Khalefa. Flexible and extensible preference evaluation in database systems. *ACM Trans. Database Syst.*, 38(3):17, 2013.

[7] X. Miao, Y. Gao, B. Zheng, G. Chen, and H. Cui. Top-*k* dominating queries on incomplete data. *IEEE Trans. Knowl. Data Eng.*, 28(1):252–266, 2016.