Skyline Queries Based on User Locations and Preferences for Making Location-Based Recommendations

Kazuki Kodama† Yuichi Iijima† Xi Guo† Yoshiharu Ishikawa‡†
†Graduate School of Information Science, Nagoya University
‡Information Technology Center, Nagoya University
Nagoya, Aichi 464-8601 Japan
{kodama,iijima,guoxi}@db.itc.nagoya-u.ac.jp, ishikawa@itc.nagoya-u.ac.jp

ABSTRACT

Due to the recent development of mobile computing and communication network technologies, information services for mobile phone users and car navigation systems have becomeof some importance. Since these mobile devices have limited display sizes, we often need to select carefully the appropriate information to be presented to the user. However, it is not easy to select the "appropriate" information because users have different contexts and preferences.

In this paper, we present an approach to recommending items such as restaurants to a mobile user taking into account his current location and preferences. In our framework, a user initially provides a *profile*, which records preferences as relative orders within predefined categories such as food types and prices. We then select items to be recommended from the database based on the user's profile as well as the current location. To select good items, we extend the notion of *spatial skyline queries* to incorporate not only distance information but also categorical preference information.

Based on the proposed approach, a prototype system has been implemented in a small mobile PC containing a small embedded RDBMS. The facilities of the RDBMS, such as spatial indexes, were used to process our skyline queries effectively.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

General Terms

Algorithms, Design, Management

Keywords

Spatial databases, mobile databases, skyline queries, preferences, recommendation, location-based services

ACM LBSN '09, November 3, 2009. Seattle, WA, USA

1. INTRODUCTION

Progress in the development of mobile device technologies, communication technologies, and GPS systems in recent years has enabled the provision of rich information services for users with changing contexts and locations. Since the amounts of storage and processing power of mobile devices are now quite substantial, it has become easy to embed small databases in mobile devices for providing useful services and information to users. Therefore, a promising scenario for enabling a mobile information service based on current technology would be to select appropriate information from such a database based on the user's situation and preferences and use this information to make a recommendation.

Although the amount of data that can be handled is increasing, it is still difficult to browse and examine a large amount of data using a mobile device. Reasons for this include the limited display size of such a device and the lack of time a user to examine a large amount of data while moving. Carefully selected information is often favored because it can be easily understood. Thus, it becomes important to select a limited number of items to be recommended to the user.

Currently, information recommendation is an important research topic in many fields of computer science, including web site design, e-commerce, and digital libraries [1]. Recommendation for mobile users is an important sub-topic especially for location-based services [5]. The most common approach is to consider the distances from the user to the target items such as restaurants, with the nearest items being selected as most appropriate. There are many options for computing distances: the Euclidean distance, the distance in a road network, and a sophisticated adaptive distance [4]. However, the distance-based approach has a problem: the nearest items may not satisfy the user's preferences. Target items such as restaurants have several features such as food types and price ranges. When we recommend items to a mobile user, we need to consider not only distances but also the user's preferences about such features.

We therefore propose an approach to recommending neighborhood items such as restaurants and shops based on both the location and the preferences of a user. We extend the notion of *skyline queries* [2] for selecting a small number of "good" items from a database. In our approach, we consider two factors for selecting items:

- The distance between each item and the user, and
- The preferences of the user specified in a *profile*.

A profile specifies the relative order of the domain values for each categorical attribute in the target database. Profiles will be described in more detail later. Although there exist some proposals for *spatial queries* based on distances [6, 8], our approach is novel because we consider preference information in addition to distances. Moreover, we further extend the query processing algorithm so that at least k good items are extracted (where k is a number specified by the user) even when the original skyline contains fewer than k items.

We have constructed a prototype recommendation system by implementing our algorithms in a mobile PC with a small embedded RDBMS. The use of an RDBMS enables us to manipulate a large amount of data with a high-level interface and to process queries efficiently using its spatial index facility. The prototype system has some additional features such as ranking based on an additional condition specified by the user. We demonstrate a restaurant recommendation system running on the prototype system.

The rest of this paper is organized as follows. In Section 2, we discuss a motivating example. Section 3 describes how queries for recommendation are performed, and we present query processing algorithms. In Section 4, we explain the system framework, describing the features of the underlying RDBMS and the system architecture. Section 5 evaluates the outputs of the prototype system based on different scenarios. In Section 6, we describe related work. Finally, in Section 7, we present our conclusions.

2. A MOTIVATING EXAMPLE

Let us consider the task of providing recommended restaurant information to mobile users. The example is referred to throughout the paper.

A restaurant database is managed by the RDBMS in the user's mobile device. Figure 1 shows an example restaurant database. The database has a *Location* attribute, which stores the two-dimensional coordinates of restaurants. In addition, the database has two other categorical fields: *Type* and *Price*.

Restaurant	Location	Type	Price
a	(3, 9)	Chinese	Low
b	(7, 5)	French	Medium
c	(7, 7)	Japanese	Low
d	(5, 1)	Italian	High
e	(3, 4)	Japanese	Medium
f	(4, 8)	French	Low
g	(5, 6)	Chinese	High
h	(1, 3)	Italian	Low
i	(5, 2)	Japanese	Medium
j	(9, 3)	Chinese	High

Figure 1: Restaurant Database

Let us assume that the user's current location is (5, 5). In Fig. 2, we show the user's location and the locations of the restaurants.

In our system, we assume that a user provides a *profile* which describes the users' preference information before he or she starts moving. For example, suppose a user Alice has the following preferences:

1. She likes Italian food best. She does not want to go to a Chinese restaurant today because she had Chinese food earlier for lunch. Japanese food and French food are not bad, but she cannot decide which is better. This means that there exist preference relationships between Type values: Italian $\prec$ Japanese $\simeq$ French $\prec$ Chinese. The symbol '$\simeq$' indicates a tie (i.e., indifference) between two items.

2. Higher prices are preferable because she wants to celebrate her birthday. These preferences can be expressed as an order between Price values: High $\prec$ Medium $\prec$ Low.

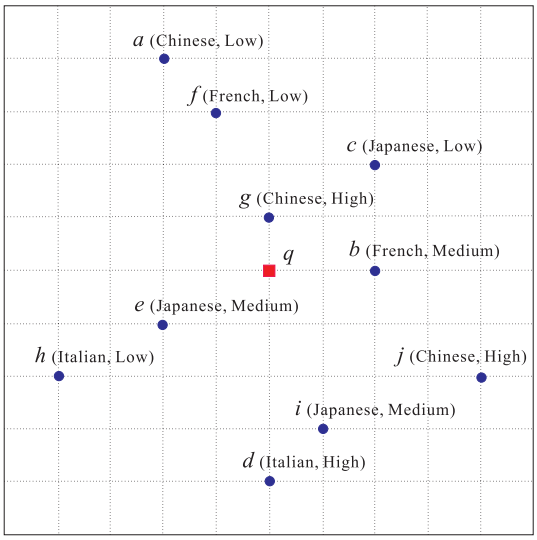

Figure 2: Restaurant Locations

In our framework, we assume that such preference information (preference orders) is stored in a mobile device as the user's *profile*. A profile should satisfy the following two conditions:

- it should cover all categorical attributes (Type and Price in this example), and
- for each category attribute, an *order* between category values (possibly with ties) is specified by the user.

We assume that a user can specify a *total order* for each categorical attribute in most cases. This would not be a complicated task for a categorical attribute with a small domain size. The user may also specify a partial order. For example, the preferences {Italian $\prec$ Japanese, Italian $\prec$ French, Japanese $\prec$ Chinese, and French $\prec$ Chinese} look similar to the example above, but the difference is that there is no preference relationship between French and Japanese. Our algorithm, shown later, can also deal with partial orders but the number of skyline objects will increase in general.

In addition to the categorical attributes, we assume the existence of a *Location* attribute that stores the location of each item which is included in the database. We can also consider the existence of other non-categorical attributes. For example, the database can have a *Score* attribute which contains the evaluation score for each restaurant, where the score is a real number in the interval [0.0, 10.0]. If the domain values of an attribute have a total order (larger is better or smaller is better), we can easily incorporate such an attribute in our algorithms. However, we omit non-categorical

attributes in the following discussion to simplify the presentation.

Let us consider what the result set would be for the example shown above. Obviously, restaurants b and g are in the result set. Restaurant d is also in the result set because its Type and Price attribute values are good although it is not close to the user. In contrast, a and f can be omitted as they are far from the user and the user does not like their categorical values. Restaurants c and e are not bad due to their closeness, but b is actually better than them. We can easily see that i, h, and j are not good items.

Using the query processing algorithm described later, we can find the "good" restaurants (skyline objects) b, d, and g for this case.

3. QUERY PROCESSING

3.1 Skyline Queries

A *skyline query* is an approach to select a limited number of "good" items from a large database [2], and there are many proposals for implementing this approach. For example, some papers focus on processing *spatial skylines* [6, 8], which consider the skyline query problem in the context of spatial databases. In short, a *skyline object* is an object which is superior to other objects at least in terms of the combination of some attributes. The set of all the skyline objects in a database is called the *skyline*, and a query to find a skyline is called a *skyline query*.

We now formally define these concepts. Let the set of all objects be $\mathcal{O}$. Each object o ($o \in \mathcal{O}$) has an ID, a location (a spatial coordinate $l = (x, y)$), and a value for each attribute $a \in \mathcal{A}$, where $\mathcal{A}$ is the set of other attributes. For simplicity we assume that all attributes in $\mathcal{A}$ are *category attributes*. In the example of Section 2, Type and Price are category attributes. In the following, we denote the coordinates of object o by $o.l$, and $o.a$ represents the value of attribute a ($a \in \mathcal{A}$) for object o.

An important notion in skyline queries is that of a *dominance relationship* between objects. To define the dominance relationship, we assume that there is a total order $\prec$ for each attribute. When $o.a \prec o'.a$, we say that o is superior to o' in terms of attribute a. The definitions of a dominance relationship and a skyline query are as follows.

Definition 1 (Dominance Relationship). *Given two objects o and o', if o is equal to or better than o' in terms of all attributes, and if o is better than o' at least one attribute, we say o* dominates *o', and write $o \prec o'$. Formally, we have*

$$o \prec o' : (\forall a \in \mathcal{A}^+, o.a \preceq o'.a) \wedge (\exists a \in \mathcal{A}^+, o.a \prec o'.a),$$

where $\mathcal{A}^+$ is the union of all the categorical attributes $\mathcal{A}$ and the location attribute l ($\mathcal{A}^+ = \mathcal{A} \cup \{l\}$). The precedence relationship for the location is based on the Euclidean distance.

Definition 2 (Skyline Query). *A* skyline query *is a query to select the set S of all the objects such that are not dominated by other objects. S is called the* skyline *and formally given as follows:*

$$S(O) = \{o \in O \mid \neg\exists o' \in (O - \{o\}) \wedge o' \prec o\}.$$

3.2 Preference-Based Query Processing

In our work, we extend the notion of skyline queries shown in the previous subsection and consider a variation of spatial skylines [6, 8]. We utilize the Euclidean distance and treat closer objects as candidate skyline objects. We assume that the dominance relationship for each category attribute can be obtained from the user's profile. For efficient query processing, we employ the concept of nearest neighbor queries and evaluate progressively whether each object belongs to the skyline. If an object is not dominated by other objects in terms of the distance and all the category attributes, it belongs to the skyline.

We illustrate the idea of query processing using the example discussed in Section 2. Since we check objects in order, from the nearest neighbor object to further objects, we can determine the dominance relationship by examining categorical attributes, as described later. First, we do some preprocessing. We project the table shown in Fig. 1 for category attributes Type and Price. In addition, since the equivalence relationship "Japanese $\simeq$ French" holds in the profile, we merge two attribute values. Figure 3 shows the resulting table after preprocessing. We call it the *category table*. It represents *all the combinations* of attribute values that appear in the database.

Type	Price
Italian	High
Italian	Low
Japanese/French	Medium
Japanese/French	Low
Chinese	High
Chinese	Low

Figure 3: Category Table

Next, we retrieve the nearest restaurant using a nearest neighbor query. In our case, we get restaurant g (Chinese, High) which is obviously not dominated by other restaurants in terms of distance, so it belongs to the skyline. Thus, we get $S = \{g\}$ as the intermediate state of the skyline. Then, we delete the entries which are dominated by the current skyline. In this example, restaurant g (Chinese, High) dominates the patterns (Chinese, High) and (Chinese Low). Therefore, we delete these two patterns. Step (1) in Fig. 4 shows the transition, where IT, JP, FR, and CN correspond to Italian, Japanese, French, and Chinese, respectively, and H, M, N represent High, Medium, and Low. After Step (1), we have a category table with four entries. A restaurant whose pattern is not contained in this table does not belong to the skyline.

Type	Price			Type	Price			Type	Price
IT	H								
IT	L	(1)		IT	H	(2)		IT	H
JP/FR	M	$\Longrightarrow$		IT	L	$\Longrightarrow$		IT	L
JP/FR	L			JP/FR	M	×			
CN	High	×		JP/FR	L	×			
CN	Low	×							

Figure 4: Transition of Category Table

Let us continue the example. We now retrieve the secondary nearest neighbor object. In this case, we get restaurant b. Since b's attribute pair (French, Medium) is contained in the current category table, we know that b is not dominated by the current skyline. Thus, b belongs to the skyline and we get $S = \{g, b\}$. Then we delete the patterns (Japanese/French, Medium) and (Japanese/French, Low)

from the category table as shown in Step (2) of Fig. 4.

Continuing the nearest neighbor query, we get restaurant e. Since the attribute pair (Japanese, Medium) is not in the category table, we know that e is already dominated by other objects and, therefore, e is not a skyline object. As we continue the process, we find that c (Japanese, Low), f (French, Low), and i (Japanese, Medium) are already dominated and not skyline objects. After retrieving d (Italian, High), we know that d belongs to the skyline ($S = \{g, b, d\}$) because its attribute pair is in the table. Then we delete the two entries (Italian, High) and (Italian, Low), which are dominated by (Italian, High), from the table. Then the category table is now empty. This means that even if we continue the process, the target objects never belong to the skyline. Thus, we can quit the process, and the resulting S is output as the skyline.

Based on the above example, we formalize the algorithm in Algorithm 1. In the algorithm, S represents the resulting set of skyline objects. The function get_category_table() creates the initial category table from the user's profile. The function NN_init(q) initializes the underlying nearest neighbor query using the query point q (we assume that we can use a spatial index for this purpose). After that, we can get objects from the nearest one by repeated calls of NN_fetch(). The Boolean function match(T, o) tries to find an entry e of T such that the categorical attribute values of o exactly match with e. At lines 11 and 12, we delete the entries dominated by o from the category table T. The repeating loop ends when T becomes empty.

Algorithm 1 Spatial Skyline with Preferences

```
1:  function SSP(p, q)
2:                          ▷ p: user's profile, q: user's location
3:      S ← ∅                                          ▷ Skyline objects
4:      T ← get_category_table(p)
5:      NN_init(q)          ▷ Initialize nearest neighbor query
6:      repeat
7:          o ← NN_fetch( )          ▷ Get next nearest object
8:          if match(T, o) then
9:                                        ▷ o is a skyline object
10:             S ← S ∪ {o}
11:             Remove the matched entry e ∈ T from T
12:             Remove all the entries e ∈ T such that o ≺ e
13:         end if
14:     until T is empty
15:     return S
16: end function
```

3.3 Multi-level Skyline Query

The skyline query algorithm shown in Algorithm 1 presents good items based on the dominance relationship. The algorithm works well, but it has the problem that the resulting skyline may consist of a small number of objects. This happens when one or two objects are very strong compared to other objects, or when the number of category attributes is small. For example, we only have three skyline objects in the result of the example query shown in Fig. 5. A user who wants to compare several candidates would not be satisfied by such a result.

As a solution for this problem, we propose the idea of *multi-level skyline queries*. We call the skyline objects obtained by the algorithm in the previous section the *first-level* skyline objects. If the number of the first-level skyline objects is smaller than k, which is a number specified by the user, we compute the second-level skyline objects. For this computation, we eliminate the first-level skyline objects from consideration.

Figure 5: Skyline Objects

We explain the idea using the example from Section 2. Assume that the user needs at least five objects and so specifies $k = 5$. Using the skyline query algorithm shown above, we get the first-level skyline $S_1 = \{b, d, g\}$. Since the number is smaller than the threshold k, we try to find the second-level skyline objects by eliminating the objects in S_1 from the candidates. Using the same algorithm, we get the second-level skyline objects: in this case, $S_2 = \{e, h, j\}$. Since we get six skyline objects in total, we can end the procedure here. If the total number were still smaller than k, we would continue to the third-level selection. Figure 6 shows the resulting skyline objects. A star mark and a circle represent the first-level and the second-level skyline objects, respectively.

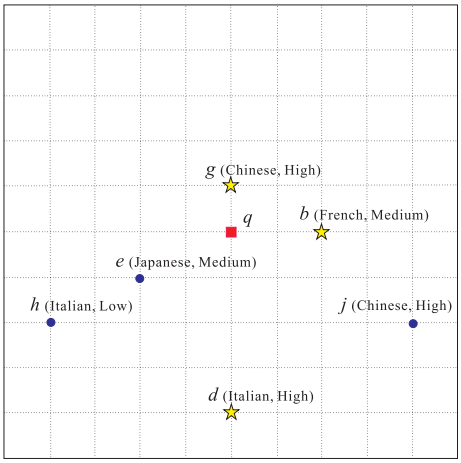

Figure 6: Multi-Level Skyline Objects

Algorithm 2 shows the multi-level spatial skyline algorithm. The algorithm is straightforward and the result S contains the sets of skyline objects for different levels ($S = \{S_1, S_2, \ldots\}$). At line 7, the algorithm calls function SSP2, which is an extended version of SSP (Algorithm 1). For

SSP2, we need to make the following modifications to Algorithm 1:

line 1: **function** SSP2(p, q, S)
line 8: **if** match$(T, o) \wedge o \notin S$ **then**

The modification in line 8 means that the algorithm finds new skyline objects and ignores existing ones S.

Algorithm 2 Multi-Level Spatial Skyline with Preferences

```
1: function MSSP(p, q, k)
2:                          ▷ k: minimum number of result objects
3:     S ← ∅
4:     count ← 0
5:     level ← 1
6:     while count < k do
7:         S_level ← SSP2(p, q, S)
8:         count ← count + |S_level|
9:         level ← level + 1
10:        S ← S ∪ {S_level}
11:    end while
12:    return S
13: end function
```

4. SYSTEM FRAMEWORK

In this section, we describe the framework for our prototype system.

4.1 Embedded RDBMS Entier

Entier [3] is provided by the Hitachi company as a compact RDBMS for embedded systems. Although multi-user transaction facilities are not supported, Entier provides standard SQL features. In addition, the system supports spatial data types and quadtree-based spatial indexes. The feature is helpful for supporting mobile applications and car navigation systems.

In the following, we describe some features of Entier related to our prototype. Entier applications can be written in C, C++, or Java. We have used C++, together with ECLI (Entier Call Level Interface) functions for searching and updating databases.

The Restaurants relation discussed in Section 2 is created as follows.

```
CREATE TABLE Restaurants (
  id int,
  name varchar(128),
  location geompoint,
  type varchar(16),
  price varchar(16)
);
```

The datatype geompoint is provided by Entier, and encodes the (x, y) coordinates of a spatial point. We can also create a spatial index for the relation as follows:

```
CREATE INDEX loc_idx
  ON Restaurants(location) EMPTY;
```

With this command, a quadtree-based index is created for the location attribute.

In addition to the above facilities, Entier supports efficient spatial query support using spatial indexes. For example, a distance-based range query is written as follows.

```
SELECT id, name
FROM Restaurants
WHERE WITHIN("location", ..., ...) = TRUE
```

In the second and third arguments of the `WITHIN` predicate we need to encode the range-query condition in a binary representation (the detail is omitted here). Unfortunately, Entier does not have a direct support for nearest neighbor queries in the version we used. Therefore, we wrote a simple wrapper to perform a nearest neighbor query using the distance-based query feature: we simply perform a range query with a predefined distance threshold then iterate over the sorted objects.

4.2 System Organization

The system organization is shown in Fig. 7. The prototype was implemented on a small Windows note PC. We used a Web browser as the interface of the system. We could have constructed a more sophisticated interface using the features of the chosen mobile device, but we decided to simplify the implementation.

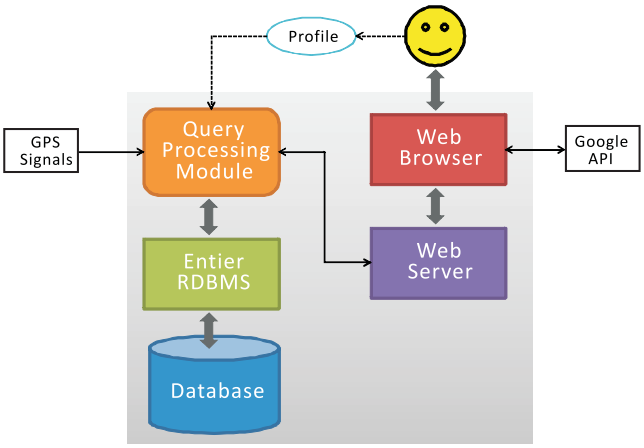

Figure 7: System Organization

A user initially specifies his preferences as a profile and sets parameter k, the minimum number of result objects, using a Web browser. The profile and the parameter are stored in the query processing module. The query processing module is the main part of our framework and performs skyline query processing. When the user starts moving, GPS signals are obtained periodically and queries are performed based on the user's current location. Some part of query processing is performed in the underlying database system Entier. The query processing module issues spatial SQL queries to Entier, if required. When skyline objects are computed, the query processing module constructs an HTML file, which is displayed on the Web browser. We used Google Map (called via the Google API) to provide a map-based interface and Ajax to implement interactive features.

5. SYSTEM EVALUATION

In this section, we evaluate the results of our prototype system. Since comparison-based evaluation is hard for our problem, we examine how the system features were implemented.

5.1 System Images

Figure 8 shows a displayed image of the resulting map interface on a Web browser. It shows the downtown area of Nagoya, Japan and the user is assumed to be in the center of the map. Selected restaurants (skyline objects) are represented by "markers". The points A, B, and C are explained in the following subsection.

Figure 8: System Image (Profile P1, Location A)

If we click a displayed marker, we can see the detailed information for the restaurant. Figure 9 shows an example of this feature.

Figure 9: Displaying Detailed Information. The Japanese text displayed means (Restaurant Name: DANTE, Type: Italian, Price, Medium, Level: 1) in English.

5.2 Scenario-Based Experiments

We performed some scenario-based experiments. Here we briefly report the results.

The target area is the downtown area of Nagoya. We collected information about 100 restaurants in this area from a Web site[1]. As described in the former examples, the restaurant database has a Location attribute and two category attributes Type and Price. The attribute Type has four domain values: Japanese, Chinese, French, and Italian. Each type has 25 instances in the restaurant database. The attribute Price has three domain values: High, Medium, and Low. We used two sample user profiles and checked how the results change depending on the preferences. The contents of these profiles are summarized in Fig. 10.

Profile	Type	Price
P1	IT $\prec$ JP $\simeq$ FR $\prec$ CN	H $\prec$ M $\prec$ L
P2	JP $\prec$ CN $\prec$ IT $\prec$ FR	L $\prec$ M $\prec$ H

Figure 10: Sample Profiles

We explain the system behavior based on an example scenario using the location A, B, and C shown in Fig. 8. The user starts from A and goes to C via B. We assume that the user examines the current skylines at each place. For profile P1, the query result at A is shown in Fig. 8. The result changes when the user reaches B as shown in Fig. 11. Figure 12 shows the skyline objects at C for profile P1. Since locations B and C are close, their results are quite similar.

Figure 11: Skyline Objects (Profile P1, Location B)

For skyline computation, we employed two different threshold values $k = 10$ (default) and $k = 5$. The results are summarized in Table 1. The columns "Level 1", "Level 2", and "Level 3" report the number of objects in each skyline level for the default case $k = 10$. For profile P2, the query results at locations B and C do not have level 3 results because the threshold $k = 10$ is reached at level 2. The "Elapsed Time (ms; $k = 10$)" columns show the processing (wall clock) time in milliseconds for the case $k = 10$ (default) and $k = 5$.

From the table, we can see that query processing is quite fast and a user will not become frustrated during interactive browsing. Of course, this is partly due to the fact that we only used 100 restaurants in the experiments; the query response time will increase when the number of restaurants is increased. However, we estimate that the response time will

[1]http://www.gnavi.co.jp/

Table 1: Experimental Results

Profile	Location	Level 1	Level 2	Level 3	Elapsed Time (ms; $k = 10$)	Elapsed Time (ms; $k = 5$)
P1	A	3	3	9	172	47
	B	3	6	4	157	78
	C	3	6	4	157	78
P2	A	3	6	8	187	78
	B	4	6	—	125	94
	C	4	6	—	125	94

Figure 12: Skyline Objects (Profile P1, Location C)

not increase much. The first reason for this is that the density of restaurants will not increase even if we cover a larger area and the database becomes huge. The second reason is that we do not have to consider restaurants located far from the user's position due to the existence of a spatial index.

6. RELATED WORK

Skyline queries [2] are used for selecting a limited number of interesting items from a large database, and the topic is currently being intensively studied. Many skyline query algorithms have been proposed for different types of databases and different situations.

The topic most related to our work is the *spatial skyline query*. Zheng et al. [8] propose a query processing method to produce spatial skylines for location-based services. They focus on location-dependent spatial queries (LDSQ) and consider a continually changing user location (query point). In this situation, it is not easy to decide how often we should update the current skyline presented to the user. If the update period is long, we may fail to catch a change of skyline objects. On the other hand, if the update period is too short, the query processing cost becomes high. Zheng et al. [8] introduce the idea of a *valid scope*, which is a region in space. While a moving user is still located in the valid scope, the skyline query returns the same result so that the system does not have to issue a new query. Although our current algorithm does not use this technique, we might be able to enhance the algorithms to incorporate the idea.

Sharifzadeh and Shahabi [6] present another type of spatial skyline problem. In their scenario, the query is issued by a group of users. They are located in different positions and want to find nearby facilities. Support for multiple users is not considered in our current system, but it would be an interesting direction for future research.

Wong et al. [7] propose a query processing method to find skyline objects in a database that includes *nominal attributes*. Typically, a nominal attribute (categorical attribute in our terminology) does not have an inherent order and the order between the domain values depends on each user. For a database with many nominal attributes, the paper proposes the construction of a kind of index (called a TPO-tree) which summarizes the order structure for the user. Although they consider categorical attributes with orders specified by users, they do not consider dynamic distances for computing skylines. We might be able to extend our algorithms to use such additional data structures.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed an approach to recommending items to a mobile phone user taking into account his current location and preferences. The main feature is that we consider not only the distances between items and the user, but also preference orders for categorical attributes. Such preferences are user-dependent and represented by a profile for the user. The query processing algorithm we propose is an extended version of spatial skylines. In addition, we have extended the algorithm to output at least k good items, where k is a threshold specified by the user.

A prototype system was implemented using Entier, a small embedded relational database system. This RDBMS supports SQL-based database manipulation together with spatial query and spatial index facilities. As an example application, we created a restaurant recommendation system. It considers the user's current position and his preferences for food type and price range. We performed some experiments using real datasets and evaluated the behavior of the system and its query processing performance.

Apart from future work based on ideas suggested by related research (see previous section), we are considering possible improvements to the way results are displayed. The current system displays all the skyline objects together. To make the system easier-to-understand, we could present skyline objects incrementally based on the level: first, the system presents the level-1 objects; if the user is not satisfied, the system displays additional level-2 objects, and so on. In this case, the user does not need to specify parameter k since he can request the next level if he is not satisfied with the current result. Another idea is to present dominance re-

lationships between objects graphically on the display. For example, we can display "arrows" from a level-1 object to the level-2 objects it dominates. It is not difficult to display the reason why a level-2 object is worse than the level-1 object when the arrow between them is clicked. We are also planning to enhance the system to incorporate other types of preference information, such as user movement histories and interaction with other users.

8. ACKNOWLEDGMENTS

This research was partly supported by a Grant-in-Aid for Scientific Research, Japan (19300027, 21013023).

9. REFERENCES

[1] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TKDE*, 17(6):734–749, 2005.

[2] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proc. ICDE*, pages 421–430, 2001.

[3] Embedded database Entier (in Japanese). http://www.hitachi.co.jp/Prod/comp/soft1/Entier/.

[4] Yoshiharu Ishikawa, Hiroyuki Kitagawa, and Tooru Kawashima. Continual neighborhood tracking for moving objects using adaptive distances. In *Proc. IDEAS*, pages 54–63, Edmonton, Canada, 2002.

[5] Jochen Schiller and Agnès Voisard. *Location-Based Services*. Morgan Kaufmann, 2004.

[6] Mehdi Sharifzadeh and Cyrus Shahabi. The spatial skyline queries. In *Proc. VLDB*, pages 751–762, 2006.

[7] Raymond Chi-Wing Wong, Ada WaiChee Fu, Jian Pei, Yip Sing Ho, Tai Wong, and Yubao Liu. Efficient skyline querying with variable user preferences on nominal attributes. In *Proc. VLDB*, pages 1032–1043, 2008.

[8] Baihua Zheng, Ken C. K. Lee, and Wang-Chien Lee. Location-dependent skyline query. In *Proc. MDM*, Beijing, China, 2008.