Selecting a Good Conference Location Based on Participants' Interests

Muhammed Miah
mmiah@suno.edu
Department of Management Information Systems
Southern University at New Orleans

Abstract

Selecting a good conference location within budget constraints to attract paper authors and participants is a very difficult job for the conference organizers. A conference location is also very important along with other issues such as ranking of the conference. Selecting a bad conference location may reduce the number of paper submissions and create bad impressions on the conference to the paper authors and conference participants. The conference location should be selected in such a way that it can attract authors to submit papers as well as others to participate/attend. In this paper we discuss how to select a good conference location within budget constraints that can attract many authors/participants considering participants' interests. We propose several methods to select the best location among the available possible locations within budget constraints based on the authors and participants interests on various features or attributes of the locations. Our problem also has interesting applications in information systems education as well. We perform evaluation of our proposed algorithms both on real and synthetic data.

Keywords: selecting conference location, budget constraints, authors and participants interests.

1. INTRODUCTION

Selecting a good location is one of the key issues in success of a conference in addition to other important aspects such as budget/cost. It is not an easy job to select a perfect location for a conference to attract and satisfy conference participants; and even harder within budget constraints. The chance of success of a conference depends in a major way upon the choice of the conference location. The location has to be selected with extreme care, in order to organize the conference successfully. As one of the most important ingredients to a successful conference, choosing the right location is a task that cannot be taken lightly.

People tend to pick places such as hotels, resorts, and conference halls to hold conferences. Numerous factors such as the time span of the conference, the number of attendants, overall environment, ambiance of the location, technology available, surrounding activities and places, travelling, transportation, accommodations, etc. have to be taken under consideration when choosing a location.

Among the possible locations, the organizers need to select the best one that can satisfy the participants (e.g., paper authors and other attendants) in terms of the facilities available in the location as well as other factors mentioned above. We will mention participants throughout the paper which represents paper authors, workshop organizers, tutorial providers, and all other attendants.

Participants tend to prefer conference location based on certain factors such as time of the year, duration of the conference, minimum travel, comfortable accommodation, expense, interesting activities and places to visit available around, and so on. Participants prefer a conference location just not for participating and listening to the presentation, but also to visit some interesting places nearby and performing other activities as well possibly with accompanying family members, friends, colleagues, and other conference attendants. So the participants can express their interests based on the available features specific to possible locations and the organizers can choose the best location that can satisfy as many participants as possible.

Consider a conference organizer wishes to select a location from the available set of possible locations, given the feature preferences of its potential participants. For example, a conference location *A* has the following elements: *close to the beach*, *WiFi available*, *restaurant on site*, *swimming pool*, and *accommodation in the same building*. Another location *B* has the following elements: *close to the mountain*, *no WiFi available*, *free local transportation*, and *accommodation may or may not be in the same building*. The potential participants can express their interests by specifying "yes" or "no" for each element – where "yes" means interested on the element and "no" means not interested or do not care. The purpose of conference organizer is to select the location *A* or *B* based on these elements to satisfy as many participants as possible. The conference organizer can collect the preferences in terms of survey based on the possible features from the participants (previous paper authors and attendants) to select the location in future. Because of the vast use of internet now-a-days, it is very easy to collect such preferences online through online surveys, search queries, on-site surveys during conference, and other ways.

The problem also has interesting applications in information systems education such as designing an information systems course that can attract students and meets industry demands, designing a program to meet constantly changing technological world to produce better graduates, and so on.

We summarize our major contributions next.

Major Contributions:

1. We define the problem of selecting best conference location among the available possible locations within budget constraints based on participants' interests.
2. We present several algorithms based on different semantics.
3. We perform evaluation of our proposed algorithms both on real and synthetic data.

The rest of the paper is organized as follows. Section 2 provides formal problem definitions. Section 3 discusses details of the proposed algorithms. In Section 4 we present the result of extensive experiments, and discuss other interesting variants in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2. PROBLEM FRAMEWORK

In this section we formally define the main problem for Boolean data. As we discuss in Section 5, many other variants can be reduced to this problem. First we provide some useful definitions.

Available Locations Database: Let $D = \{t_1 \ldots t_N\}$ be a collection of Boolean tuples over the attribute set $A = \{a_1 \ldots a_M\}$, where each tuple *t* is a bit-vector where a 0 implies the absence of a feature and a 1 implies the presence of a feature. A tuple *t* may also be considered as a subset of *A*, where an attribute belongs to *t* if its value in the bit-vector is 1. Each tuple *t* in database *D* represents an available possible conference location.

Survey Log: Let $Q = \{q_1 \ldots q_S\}$ be collection of survey results where each tuple *q* defines a subset of attributes which represents survey result from a respondent (participant).

The problem definition is as follows:

Conference Location Selection (CLS) Problem: *Given an available locations database D, and a survey log Q, select a tuple t from D such that the number of tuples in Q satisfied by t is maximized.*

The following running example will be used throughout the paper to illustrate various concepts.

EXAMPLE 1: *Consider a database of possible available conference locations, which contains a single database table D with N rows and M attributes where each tuple represents a possible location. The table has numerous attributes that describe details of the location: Boolean attributes such as On the Beach, WiFi Available, On-site Accommodation, Close to Major International Airport, Close to National Park, National Museum in the Area, etc; numeric attributes such as Distance from the Airport, Number of Accommodations available, etc; and text attributes such as Reviews, and so on. Figure 1 illustrates such a database (where only the Boolean attributes are shown)* *of four locations available within the budget constraints. The figure also illustrates a survey log of five tuples collected from five respondents or participants. Now the job is to select a tuple t from database D that can satisfy as many tuples from survey log Q as possible* □

Location	*Beach*	*WiFi*	*Accommodation*	*Intl. Airport*
t_1	1	0	1	0
t_2	0	1	0	1
t_3	0	1	1	1
t_4	1	0	1	1

Available Locations Database *D*

Tuple ID	*Beach*	*WiFi*	*Accommodation*	*Intl. Airport*
q_1	1	1	0	1
q_2	0	1	1	0
q_3	1	1	1	0
q_4	0	1	1	1
q_5	1	0	0	1

Survey Log *Q*

Figure 1. Illustrating EXAMPLE 1

3. PROPOSED ALGORITHMS

In this section we discuss our main algorithmic results. We propose four algorithms and discuss them next in detail.

Algorithm based on Maximized Features Coverage (*MFC*)

The intuition of this algorithm is that we look for a tuple in the database of available possible locations that has maximum sum of scores over all tuples in the survey log. That means, find a tuple *t* in *D* such that it satisfies as many of the conditions or features asked by the tuples in *Q* as possible. It is a best-effort problem and hence the algorithm is polynomial time algorithm. We assume that the scoring function is an aggregation of the scores of the individual attributes/features, e.g., the sum of the attribute contributions. The attribute contribution could be 1 if it is satisfied or 0 otherwise. For a text database, it could be the tf-idf weight of a keyword. The tf–idf weight (term frequency–inverse document frequency) is a statistical measure used to evaluate how important a word is to a document in a collection or corpus, often used in information retrieval and text mining.

So the algorithm is as follows:

1. First we need to collect the available possible conference locations within the budget constraints.
2. We also collect the response on the possible features of the conference locations from the participants in the form of online/onsite survey or other ways.
3. Then for each possible available location, we see how many of the cumulative features are satisfied in the survey log by the location.
4. We select the location with highest number of cumulative features satisfied by it.

Figure 2 displays the pseudocode of the algorithm *MFC*.

```
Algorithm: MFC

Let D be the Boolean database of possible
available locations; Q be the survey log, A
($a_1…a_M$) be the attributes in D and Q

For each tuple $t_j$ in D
    int count, total = 0;
    For (int i = 1 to M)  //for each attribute
        count = # of tuples in Q satisfied
for
            $a_i$ = 1
        total += count  //Sum count with
total
Return the tuple $t_j$ with maximum total
```

Figure 2. Pseudocode of Algorithm *MFC*

Consider the algorithm *MFC* on the EXAMPLE 1 in Figure 1. The algorithm needs to select a tuple *t* from *D* that satisfies as many conditions or features asked by the tuples in *Q*. For tuple t_1, we can see that it satisfies total 6 cumulative tuples in *Q* as follows: 3 (q_1, q_3, q_5) for attribute/feature *Beach*, 0 for feature *WiFi*, 3 for feature *Accommodation* (q_2, q_3, q_4), 0 for feature *Intl. Airport*; the total (3+0+3+0) = 6 tuples. Similarly the tuple t_2 satisfies total 7 cumulative tuples in *Q* (3 for feature *WiFi* and 3 for feature *Intl. Airport*), tuple t_3 satisfies total 10 cumulative tuples in Q (3 for feature *WiFi*, 3 for feature *Accommodation*, and 3 for feature *Intl. Airport*), and tuple t_4 staisfies total 9 cumulative tuples in *Q* (3 for features *Beach*, 3 for feature *Accommodation*, and 3 for feature *Intl. Airport*). So the tuple t_3 covers maximum number of cumulative features (10) asked by the tuples in *Q*, so the algorithm *MFC* selects the location t_3 as the best location among the four locations (tuples) available in the database *D*.

Algorithm MFC with Budget Constraints: If we have a predefined budget limit in advance, we can eliminate the locations that do not meet the budget limit and simply employ the algorithm *MFC* as discussed above to select the best location. In case if we do not have a fixed predefined budget limit and want to maximize the participants' satisfaction as well as minimize the cost, the algorithm *MFC* can be employed to tackle the budget constraints as follows:

1. For each available location, algorithm *MFC* calculates a score as *the total number of features or attributes covered by the location* divided by *the total cost of all features the location provides*.
2. Select the location with the highest score. In this way, we are considering both the number of features covered and the total cost of a location and maximizing the features covered as well as minimizing the cost.

Algorithm based on Weighted Maximized Features Coverage (*WMFC*)

This algorithm is for the weighted version of the problem Conference Location Selection (*CLS)* described earlier. When participants respond to a survey and specify the features they like regarding to a specific location, sometimes they also want to mention the preference on each feature they select. A participant might prefer one feature over another and not the same preference for all the features. So the survey can be conducted with option for the participants to mention the weight for each feature selected and the sum of the weights for all the features a participant selects must be equal to one. In this situation, instead of simply counting the total number of features; we need to consider the weight on each feature given by the survey participants. Figure 3 illustrates a survey log where five participants mention the weights for each attribute/feature they like in terms of weight.

Tuple ID	*Beach*	*WiFi*	*Accommodation*	*Intl. Airport*
q_1	.5	.4	0	.1
q_2	0	.4	.6	0
q_3	.4	.4	.2	0
q_4	0	.2	.4	.4
q_5	.7	0	0	.3

Survey Log *Q′*

Figure 3. Survey log based on feature weight

As we can see in Figure 3, the sum of weight for each row (tuple) is equal to 1, that means a participants mention weight on each feature they like and the total weight must be equal to 1.

The algorithm *WMFC* is as follows:

1. First we need to collect the available possible conference locations within the budget constraints.
2. We also collect the response on the possible features of the conference locations from the participants in the form of online/onsite survey or other ways. The response on each feature represents the weight mentioned by the participants.
3. Then for each possible available location, we sum up the cumulative weights of features that are satisfied in the survey log by the location.
4. We select the location with highest cumulative weight of features satisfied by it.

Figure 4 displays the pseudocode of the algorithm *WMFC*.

Consider the available locations database *D* in Figure 1 and the survey log *Q′* based on weighted preference in Figure 2. The algorithm *WMFC* needs to select a tuple *t* from *D* that it satisfies as many conditions or features asked by the tuples in *Q′* based on the weighted preference. For tuple t_1, we can see that the total weights of the features satisfied by the tuples in *Q′* is as follows: total 1.6 for attribute/feature *Beach* (.5 for q_1, .4 for q_3, and .7 for q_5), 0 for *WiFi*, 1.2 for *Accommoda-*

tion, 0 for *Intl. Airport*; so the total (1.6+0+1.2+0) = 2.8. Similarly the total weight for the tuple t_2 is 2.2 (1.4 for *WiFi* and .8 for *Intl. Airport*), total weight for the tuple t_3 is 3.4 (1.4 for *WiFi*, 1.2 for *Accommodation*, and .8 for *Intl. Airport*), and total weight for tuple the t_4 is 3.6 (1.6 for *Beach*, 1.2 for *Accommodation*, and .8 for *Intl. Airport*). So the tuple t_4 covers maximum features weight asked by the tuples in *Q'*, so the algorithm *WMFC* selects the location t_4 as the best location among the four locations (tuples) available in the database *D*.

```
Algorithm: WMFC

Let D be the Boolean database of possible
available locations; Q be the survey log, A
($a_1…a_M$) be the attributes in D and Q; $w_i$ (i
= 1 to M) be the weight given for each
attribute $A_i$

For each tuple $t_j$ in D
    int local_sum, total_sum = 0;
    For (int i = 1 to M)  //for each attribute
        local_sum = sum of weights for all
            tuples in Q satisfied for $a_i$ = 1
        total_sum += local_sum
Return the tuple $t_j$ with maximum to-
tal_sum
```

Figure 4. Pseudocode of Algorithm *WMFC*

So, the algorithm *MFC* is modified to *WMFC* by summing up the cumulative weights on the features instead of just counting them. The two algorithms are basically the same and *WMFC* can also be used for Boolean data (survey log) where the values or weight of each feature is either 1 or 0.

Algorithm WMFC with Budget Constraints: As discussed for algorithm *MFC*, the algorithm *WMFC* also can be employed to tackle the budget constraint by calculating score for each available location as *the sum of cumulative weights a location can satisfy* divided *the total cost of all features the location provides*. Then select the location with the highest score.

Algorithm based on Survey-Specific Scoring function (*SSF*)

We consider Top-*k* Retrieval via Survey-Specific Scoring Function. Let *Score*(*q*, *t*) be a scoring function that returns a real-valued score for any tuple *t*. Let *k* (=1) is an integer associated with a survey response *q*. Then *R*(*q*) is defined as the set of top-*k* tuples in the database with the highest scores. In our problem, *k* is equal to 1 as we try to select the best one location among the available possible locations. Note that tuples that do not satisfy all attributes specified in the query may also be returned. An example of a query specific scoring function is the dot product of *q* and *t*.

Tuple ID	*Top-1 tuple with scores*
q_1	t_2 (2)
q_2	t_3 (2)
q_3	t_1 (2)
q_4	t_3 (3)
q_5	t_4 (2)

Figure 5. Results of Top-*k* (*k*=1) Retrieval

Consider the EXAMPLE 1 illustrated in Figure 1. Assume that each tuple in the survey log returns the top-1 tuple (i.e., *k* = 1), where the survey-specific scoring function is the dot product between a survey response in *Q* and a tuple in *D*. Based on this scoring function, the results of the execution of the five survey responses are shown in Figure 5 (score ties have been broken arbitrarily).

```
Algorithm: SSF

Let D be the Boolean database of possible
available locations; Q be the survey log, A
($a_1…a_M$) be the attributes in D and Q;

Initialize an empty buffer B
        //that will contain top-1 tuples with
        //corresponding scores for each tuple in Q

For each tuple $q_j$ in Q
    Find top-1 tuple from D with
        corresponding score
       //score based on the survey-specific scor
       //ing function (dot product betn. Q and D)

    If top-1 tuple found for $q_j$ already
           presents in B
        Add (sum) new score with the
            existing score for the
            corresponding tuple in B
    Else
        Insert top-1 tuple found for $q_j$ in B
 Return the tuple in B with highest score
```

Figure 6. Pseudocode of Algorithm *SSF*

Once we find the top-1 tuple for each survey response, the next step of the algorithm *SSF* is to find the tuple *t* with highest cumulative scores. As we can see in Figure 5 that tuple t_3 has the highest cumulative scores of 5 (2 for q_2 plus 3 for q_4). So the algorithm *SSF* returns tuple t_3 as the best location (tuple) among the available locations in database *D*.

Figure 6 displays the pseudocode of the algorithm *SSF*.

Algorithm SSF with Budget Constraints: The algorithm *SSF* can be employed to tackle the budget constraint by calculating rank for each available location as *the score calculated by the algorithm* as in Figure 6 divided by *the total cost of all features the location provides*. Then select the location with the highest rank.

Algorithm based on Skyline Semantics Approach (*SSA*)

We also consider skyline retrieval semantics for this problem. Given a set of points, the skyline comprises the points that are not dominated by other points. A point dominates another point if it is as good or better in all dimensions and better in at least one dimension (Tan, Eng, & ooi, 2001). We consider skyline for Boolean data in our problem, but to get a clear picture let consider a common example in the literature, "*choosing a set of hotels that is closer to the beach and cheaper than any other hotel in distance and price attributes respectively from the database system of the travel agents'* (Borzsonyi, Kossmann, & Stocker, 2001)". Figure 7 illustrates this case in *2*-D space, where each point corresponds to a hotel record. The *x*-axis and *y*-axis specify the room price of a hotel and its distance to the beach respectively. Clearly, the most interesting hotels are *{a, g, i, n}*, called *skyline*, for which there is no other hotel in *{a, b, . . . , m, n}* that is better on both dimensions. As mentioned earlier, we mainly consider Boolean skylines (skylines with Boolean data), where all the attributes asked by a survey response need not to be present in the tuple to be returned by the query unlike conjunctive Boolean retrieval. Consider our running example in Figure 1. Tuple q_1 in the survey log *Q* asks for the features *Beach*, *WiFi*, and *Intl. Airport*. The tuples t_2 (for features *WiFi* and *Intl. Airport*), t_3 (for features *WiFi* and *Intl. Airport*), and t_4 (for features *Beach* and *Intl. Airport*) from database *D* would appear in the skyline as there is no tuple in *D* that exactly satisfies the conditions or features asked by q_1.

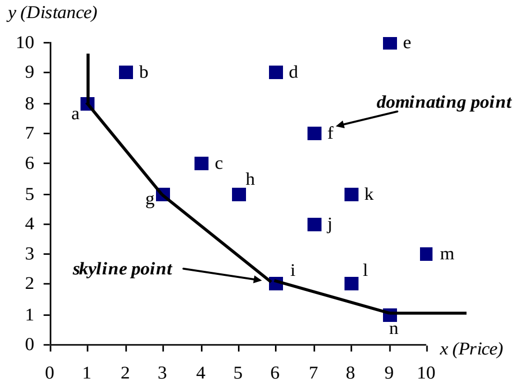

Figure 7. Skyline Example

The algorithm *SSA* works as follows:

1. We collect the available locations data base *D* and the survey log *Q*
2. Then we find the skylines (tuples for *D*) for each of the survey tuples from the survey log *Q*
3. Return the tuple in *D* that appears in most of the skylines. The tie is broken arbitrarily.

For each survey tuple *q* in the query log we define the *survey skyline* $S(q) = \{s_1 \ldots s_L\}$, which is a collection of *skyline points.* Each skyline point *s* defines a subset (i.e., projection) of attributes for which any data point (tuple) remains on the skyline. We store the data tuples from database that appear on the skylines in skyline log for each survey tuple. A skylines log contains all the skylines for the survey log. Figure 8 displays the skyline log for our EXAMPLE 1 described in Figure 1. There are several methods proposed for efficient processing of skyline queries which are mentioned in related work (Section 6). Any good skyline processing technique such as (Morse, Patel, & Jagadish, 2001) can be used here to find the skylines for the survey log which is efficient for Boolean data. Once these skylines have been found, the next step of the algorithm *SSA* is to return the tuple that appears in highest number of skylines.

Tuple ID	*Skylines (data tuples in the skyline)*
q_1	t_2, t_3, t_4
q_2	t_3
q_3	t_1 , t_3, t_4
q_4	t_3
q_5	t_4

Figure 8. Skyline Log

From Figure 8, we can see that the tuple t_3 appears in highest 4 skylines (for q_1, q_2, q_3, and q_4). So the algorithm *SSA* selects the location t_3 as the best location among the four locations (tuples) available in the database *D*.

Figure 9 displays the pseudocode of the algorithm *SSA*.

```
Algorithm: SSA

Let D be the Boolean database of possible
available locations; Q be the survey log, S
be the skyline log for Q and D;

Find skyline log S
        // skylines for Q and D

Return the tuple that appears in highest
    number of skylines in S
```

Figure 9. Pseudocode of Algorithm *SSA*

Algorithm SSA with Budget Constraints: The algorithm *SSA* can be employed to tackle the budget constraint by calculating score for each available location as *the number of skylines it appears on* as described above (Figure 9) divided by *the total cost of all features the location provides*. Then select the location with the highest score.

4. EXPERIMENTS

Our main performance indicator is the time cost of the proposed algorithms. As algorithm *WMFC* and *MFC* are basically same, we do not show the experiment results for *WMFC*. We evaluate the time performance of three algorithms *MFC*, *SSF*, and *SSA*. We do not provide any evaluation on quality as each of the proposed algorithm is using different semantics and hence is not possible to compare them with any single optimal answer. It is up to the organizers how they want to satisfy the potential conference participants. But as mentioned above, we evaluate their time performance.

System Configuration: We used Microsoft SQL Server 2000 RDBMS on a Intel Core i7 P4 2.93-GHZ PC with 3 GB of RAM and 700 GB HDD for our experiments. Algorithms are implemented in C#.

Datasets: We used both real and synthetic data for our experiments. We randomly selected five (5) available possible locations and selected 30 possible Boolean features/attributes related to these locations such as *On-site accommodation*, *WiFi available*, *Close to international airport*, and so on. We then generated a survey with the same 30 Boolean attributes to collect data from the potential participants to express their interests on the feature/attributes level. In specific, we use two datasets: (i) *REAL*: real survey log, and (ii) *SYNTH*: synthetic survey log generated from the real survey log.

Real survey log (REAL): We collected 230 survey responses for possible future conference location from university users and friends through an online survey. The survey was designed with 30 Boolean features such as *On-site accommodation*, *WiFi available*, *Close to international airport*, and so on. Users were asked to select the features they prefer to have (positive) available in the possible conference location. The value of each feature/attribute selected was set as 1 and rest of the values as 0. Users selected 4-6 features on average. *WiFi available* and *On-site accommodation* were the most popular features.

Synthetic survey log generated from real survey log (SYNTH): As the real survey log is very small, it is inappropriate for scalability experiments. So we generated larger datasets from the real query log. A total of 100,000 survey responses were generated as follows: at each step we randomly select a survey response from the REAL survey log, randomly select two of its attributes and swap their values (1 to 0 and vice versa).

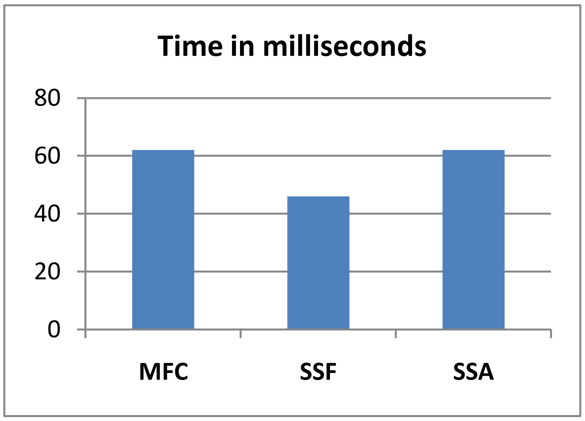

Figure 10. Time performance of the algorithms for *REAL* dataset

Figure 10 shows the time performance of the three algorithms (*MFC*, *SSF*, *SSA*) for the *REAL* dataset. The *x*-axis represents the algorithms and *y*-axis the represents the total time (in milliseconds) they take. As we can see that even the algorithm *SSF* is little slower than the other two algorithms, all the three algorithms are really very close in terms of performance. As mentioned above that the *REAL* dataset is very small with only 230 survey responses, it is not feasible to compare the performance of the algorithms. So we also conduct experiment on larger *SYNTH* dataset discussed next.

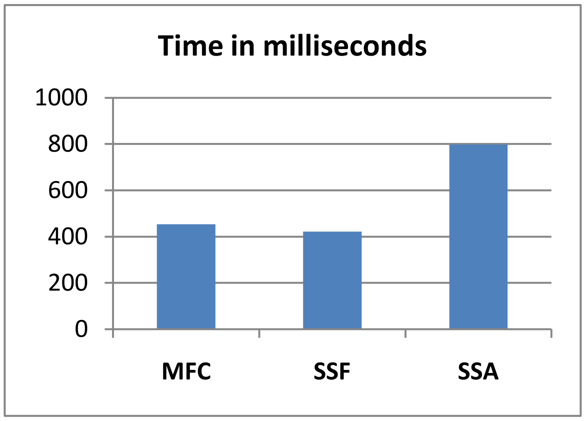

Figure 11. Time performance of the algorithms for *SYNTH* dataset

Figure 11 shows the time performance of the three algorithms for the *SYNTH* dataset. The *x*-axis represents the algorithms and the *y*-axis the represents the total time (in milliseconds) they take. As we can see that the algorithm *SSA* is little slower than the other two (*MFC* and *SSF*) and algorithms *MFC* and *SSF* take almost same time to run the experiment. The algorithm *SSA* is slower because we did not use any advanced algorithm to generate skyline log (skylines of each survey response). For this experiment we use the naïve approach comparing each tuple in the locations database for each survey response in the survey log to generate the skyline log. As mentioned in Section 4, the performance of the algorithm *SSA* can be improved by applying any effective technique to generate the skylines such as technique prposed by Morse, Patel, & Jagadish (2001).

The time performances in Figures 10 and 11 are shown in milliseconds and we can see that in fact there is not much difference in the performance of the algorithms. So, any of the algorithms can be used. But one thing to remember that each of these algorithms uses different semantics such as – algorithm *MFC* uses maximum cumulative coverage of features among all the survey responses, algorithm *SSF* uses survey-specific scoring function, and algorithm *SSA* uses skylines semantics approach. So the algorithms can select different locations based on the semantics used. In our experiment, among the available 5 locations (numbered 1, 2, 3, 4, 5), the algorithms *MFC* and *SSA* selected location number 1 as the best location whereas the algorithm *SSF* selected location number 4 as the best location. Now, it is up to the organizers which algorithm they want to use based on how they want to satisfy the potential conference participants.

5. OTHER PROBLEM VARIANTS

In this section we discuss some other interesting problem variants.

Problem Variant with Categorical Data

We consider *categorical databases*, which are natural extensions of Boolean databases where each attribute a_i can take one of several values from a multi-valued categorical domain Dom_i. A survey over a categorical database is a set of features of the form $a_i = x_i$, $x_i \in Dom_i$. We can define problem variants for categorical data corresponding to the ones for Boolean data discussed earlier.

Each categorical column a_i can be replaced by $|Dom_i|$ Boolean columns, and consequently a categorical database/survey log with *M* attributes is replaced by a Boolean database/survey log with $\prod_{1 \le i \le M} Dom_i$ Boolean attributes.

Problem Variant with Numeric Data

We also consider *numeric databases*. We consider surveys that specify ranges over a subset of attributes. The above problem variants for Boolean data have corresponding versions for numeric databases. For example, features may be specified with ranges on *price*, *distance from the airport*, *number of on-site accommodations available*, etc, and the returned results may be ranked by *price*.

Problems involving numeric ranges can be reduced to Boolean problem instances as follows. We first execute each survey response in the survey log, and reduce *Q* to *Q″* by eliminating survey response for which the new tuple has no chance of entering into the top-*k* results. Then, for each numeric attribute a_i in *Q″*, we replace it by a Boolean attribute b_i as follows: if the i^{th} range condition of tuple *q* in *Q″* contains the i^{th} value of tuple *t* in locations database *D*, then assign 1 to b_i for tuple *q*, else assign 0 to b_i for tuple *q* (i.e., each survey response has effectively been reduced to a Boolean row in a Boolean survey log *Q″*). The tuple *t* in locations database *D* can be converted to a Boolean tuple consisting of all 1's.

Problem Variant with Text Data

A text database consists of a collection of documents, where each document is modeled as a bag of words as is common in Information Retrieval. Tuples or survey responses are sets of keywords, with top-*k* retrieval via query-specific scoring functions, such as the tf-idf-based BM25 scoring function (Robertson & Walker, 1994). The Boolean problem discussed above can be directly mapped to a corresponding problem for text data if we view a text database as a Boolean database with each distinct keyword considered as a Boolean attribute. All the algorithms developed for Boolean data can be used for text data. However, if we view each distinct keyword in the text corpus (or survey log) as a distinct Boolean attribute, the dimension of the Boolean database is enormous. Consequently, none of the algorithms described above might be feasible for text data. We may need to develop new effective algorithms for text data and we plan to work on this in the future. In the future, we also intend to develop more effective algorithms for other data types described above such as categorical and numerical data.

Dependencies among Features/Attributes

Another problem variant arises when there are dependencies among the features/attributes. E.g., if a location has the *WiFi* feature available, it must also have *Internet* feature. We tackle this by removing the unsatisfiable tuples (survey responses) from the survey log and using the dependencies to optimize the algorithms.

6. RELATED WORK

Optimal product design or positioning is a well studied problem in Operations Research and Marketing which seems similar to our problem. Shocker & Srinivasan (1974) first represented products and consumer preferences as points in a joint attribute space. After that, several approaches and algorithms (Albers & Brockhoff, 1977 & 1980, Albritton & McMullen, 2007, Gavish, Horsky, & Srikanth, 1983, Gruca & Klemz, 2003, Kohli & Krishnamurti, 1989) have been developed to design/position a new product. Works in this domain require direct involvement (one or two step) of consumers and users are usually shown a set of existing alternative products to choose or set preferences. Like our work, users in fact do not get to select the attributes or features they like. Also we do not show the available locations to the users/participants instead collect their preferences on possible features level. We use previous user survey logs and it is easy to collect the preferences for large number of Internet users nowadays.

We use skyline semantics in one of our proposed algorithms, *SSA*. Several techniques have been proposed for efficient skyline query processing (Borzsonyi & Stocker, 2001, Kossmann, Ramsak, & Rost, 2002, Papadias, Tao, Fu, & Seeger, 2003, Tan, Eng, & Ooi, 2001, Sarkas, Das, Koudas, & Tung, 2008). Skyline computation over low cardinality domains (Morse, Patel, & Jagadish, 2001) also considers skyline for Boolean data as well. One main difference of our work with the existing works is that our goal is not to propose a method for processing or maintaining the skylines, instead we use skylines as a semantic where a new tuple/location can be satisfied to maximum number of potential participants.

Miah, Das, Hristidis, & Mannila (2008) tackled a related problem of maximizing the visibility of an existing object by selecting a subset of its attributes to be advertised. The main problem was: given a query log with conjunctive query semantics and a new tuple, select a subset of attributes to retain for the new tuple so that it will be retrieved by the maximum number of queries. In this paper, we consider selecting a location (tuple) from the given locations, and not selecting subset of attributes/features.

7. CONCLUSIONS AND FUTURE WORK

In this work, we investigate the problem of selecting a good conference location within budget constraints considering participants' interests on the features a location might have. The goal is to satisfy as many potential participants as possible to attract them to submit papers, arrange workshops, giving tutorials, and attend the conference. We develop several effective algorithms that work well in practice as well as for large data. We evaluate the algorithms both on real and synthetic data. In the future we plan to develop effective algorithms for different data types such as text, categorical and numerical data.

8. REFERENCES

Albers S., & Brockhoff K. (1977). A procedure for new product positioning in an attribute space. *European Journal of Operational Research*, 1(4) 230-238.

Albers S., & Brockhoff K. (1980). Optimal Product Attributes in Single Choice Models. *Journal of the Operational Research Society*, 31, 647–655.

David M. Albritton, Patrick R. McMullen. (2007), Optimal product design using a colony of virtual ants. *European Journal of Operational Research*, 176(1), 498-520.

Borzsonyi S., Kossmann D, & Stocker K. (2001). The Skyline Operator. *IEEE International Conference in Data and Knowledge Engineering (ICDE)*.

Gavish B, Horsky D., & Srikanth K. (1983). An Approach to the Optimal Positioning of a New Product. *Management Science*, 29(11) 1277-1297.

Gruca, S. T., & Klemz, R. B. (2003). Optimal new product positioning: A genetic algorithm approach. *European J. of Operational Research*, 146(3), 621-633.

Kohli, R., & Krishnamurti, R. (1989). Optimal product design using conjoint analysis: Computational complexity and algorithms. *European Journal of Operational Research*, 40, 186–195.

Kossmann, D., Ramsak, F., & Rost, R. (2002). Shooting Stars in the Sky: an Online Algorithm for Skyline Queries. *Very Large Databases (VLDB)*.

Miah, M., Das, G., Hristidis, V., & Mannila, H. (2008). Standing Out in a Crowd: Selecting Attributes for Maximum Visibility. *IEEE International Conference in Data and Knowledge Engineering (ICDE)*, 356-365.

Morse, D. M., Patel, & M. J., Jagadish, H. V. (2007). Efficient Skyline Computation over Low-Cardinality Domains. *Very Large Databases (VLDB)*.

Papadias, D., Tao, Y., Fu, G., & Seeger, B. (2003). An Optimal and Progressive Algorithm for Skyline Queries. *ACM SIGMOD*.

Robertson, S. E., & Walker, S.. (1994). Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. *ACM SIGIR*.

Shocker, A. D., & Shrinivasan, V. (1974). A consumer-based methodology for the identification of new product ideas, *Management Science*, 20(6), 921-937.

Sarkas, N., Das, G., Koudas, N., & Tung, A. K. H. (2008). Categorical skylines for streaming data. *ACM SIGMOD*, 239-250

Tan, K.-Lee, Eng, P.-Kwang, & Ooi, B. C. (2001). Efficient Progressive Skyline Computation. *Very Large Databases (VLDB)*.