Mining Preferences from Superior and Inferior Examples

Bin Jiang[1] Jian Pei[1] Xuemin Lin[2] David W. Cheung[3] Jiawei Han[4]
[1]Simon Fraser University, Canada [2]The University of New South Wales, Australia
[3]The University of Hong Kong, Hong Kong [4]University of Illinois at Urbana-Champaign, USA
{bjiang, jpei}@cs.sfu.ca, lxue@cse.unsw.edu.au,
dcheung@cs.hku.hk, hanj@cs.uiuc.edu

ABSTRACT

Mining user preferences plays a critical role in many important applications such as customer relationship management (CRM), product and service recommendation, and marketing campaigns. In this paper, we identify an interesting and practical problem of mining user preferences: in a multidimensional space where the user preferences on some categorical attributes are unknown, from some superior and inferior examples provided by a user, can we learn about the user's preferences on those categorical attributes? We model the problem systematically and show that mining user preferences from superior and inferior examples is challenging. Although the problem has great potential in practice, to the best of our knowledge, it has not been explored systematically before. As the first attempt to tackle the problem, we propose a greedy method and show that our method is practical using real data sets and synthetic data sets.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms, Experimentation

Keywords

Preferences, superior examples, inferior examples, skyline

1. INTRODUCTION

Mining user preferences plays a critical role in many important applications, such as customer relationship management (CRM), product and service recommendation, and marketing campaigns. Although many existing studies have explored how to use preferences to improve service quality such as obtaining better query answering with user preferences [14, 15, 3], effectively capturing user preferences still largely remains a challenging problem.

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

In this paper, we identify an interesting and practical problem: *mining user preferences from superior and inferior examples*. To motivate the problem, consider an application scenario where a realtor learns a customer's preference and makes recommendations.

A customer's preference on realties depends on many factors, such as price, location, style, lot size, number of bedrooms, age of the realty, developer, etc. Some attributes, such as price, are numeric and come with a well accepted preference (e.g., the cheaper in price, the better). On some categorical attributes such as location, style, and developer, the preferences often vary from customer to customer. Moreover, a customer's preferences on those categorical attributes are often oblivious or may not be obtained by a realtor in a complete and explicit way.

In order to recommend realties effectively, it is important that a realtor can capture customers' preferences well. A typical scenario is that a realtor gives a list of realties as examples. Then, a customer often picks a small subset as *superior examples* which the customer wants to see. For each superior example o, according to the customer's preferences, there *does not exist* another realty o' which is as good as o in *every* aspect, and is better than o in *at least one* aspect. This condition is necessary for a superior example since, otherwise, the user should see o' instead of o.

The superior examples differ from each other in some aspects, which reflect the tradeoffs that the customer would like to consider. For example, a customer may pick two superior examples: one with a higher price in an area by the beach, and the other with a lower price in an area without a beach. The two superior examples indicate that the customer is willing to consider the tradeoff between price and the beach.

At the same time, the customer also often picks a small subset as *inferior examples* which the customer definitely does not want to see. For each inferior example o, according to the customer's preferences, there *exists at least one* realty o' which is as good as o in every aspect, and is better than o in *at least one* aspect. In other words, the customer regards o' a better choice than o.

What can the realtor learn about the customer's preferences from those superior and inferior examples? While many realtors learn customers' preferences in the above scenarios using their professional experience, in this paper, we model the problem as mining user preferences from superior and inferior examples, and develop a data mining solution so that the learning procedure can be automated.

One may wonder whether obtaining superior and inferior

examples from a user is feasible in real applications, since a user may not want to give examples by checking thousands of realties currently available in the market. We advocate that a practical solution to the user preference mining problem is the core of the *interactive learning and recommendation procedure* which can be heavily employed in e-business and many other applications.

For example, instead of asking a user to pick examples from thousands of realties in the market, a realtor can give a short-list of tens of realties. A user can indicate some superior and inferior examples in the short list. Using the preferences learnt from this short-list and the superior/inferior examples, a realtor can filter out those realties in the market definitely uninteresting to the user, and recommend those in the market definitely preferred by the user. Among the rest that whether the customer prefers or not is unknown, another short-list can be made and provided to the customer so that more superior and inferior examples can be obtained. By the interactive and iterative learning procedure, the customer can be provided more and more accurate recommendations as the preferences are learnt progressively.

Automation of the user preference mining procedure can easily find significant applications in many domains. For example, a realtor web site can be built based on the interactive and iterative user preference learning and realty recommendation procedure described above. Such recommendation services are highly feasible and useful in many web-based business applications.

Although of great potential, to the best of our knowledge, the problem of mining user preferences from superior and inferior examples has not been explored before. In this paper, we tackle the problem and make several contributions. First, we identify and model the problem systematically. Second, our theoretical problem analysis indicates that mining preferences from superior and inferior examples is challenging. Therefore, we need to develop heuristic methods that are effective in practice. Third, we develop a greedy method and show the effectiveness and the efficiency of the method using both real data sets and synthetic data sets. Last, we discuss possible extensions of our method and several interesting and promising directions for future work.

The rest of the paper is organized as follows. In Section 2, we present a formal problem definition and study the complexity of the problem. We review the related work in Section 3. A greedy method is developed in Section 4. An extensive empirical evaluation using both real data sets and synthetic data sets is reported in Section 5. Section 6 concludes the paper.

2. PROBLEM ANALYSIS

In this section, we define the problem of user preference mining, and study the complexity of the problem. Table 1 summarizes some notions frequently used in this paper. Limited by space, the proofs of the theoretical results are omitted here, but can be found in the full version of the paper [9].

2.1 Multidimensional User Preferences

Let $\mathcal{O}$ be a set of objects in a multidimensional space $\mathcal{D} = D_1 \times D_2 \times \cdots \times D_d$. $\mathcal{O}$ can be finite or infinite. A *user preference* $\prec$ is a strict partial order on $\mathcal{O}$. That is, for objects $o_1, o_2 \in \mathcal{O}$, if $o_1 \prec o_2$, the user prefers o_1 than o_2. Object o_1 is said to *dominate* o_2. We write $o_1 \preceq o_2$ if $o_1 \prec o_2$ or $o_1 = o_2$.

Notion	Meaning
$\mathcal{O}$	A set of objects.
$\mathcal{D}$	A multidimensional feature space of objects.
$\mathcal{D}_D = \{D_1, \ldots, D_{d'}\}$	The set of determined attributes in $\mathcal{D}$
$\mathcal{D}_U = \{D_{d'+1}, \ldots, D_d\}$	The set of undetermined attributes in $\mathcal{D}$
d	The dimensionality of $\mathcal{D}$.
d'	The number of determined attributes.
o, o_1, o_2, o', o''	Objects in $\mathcal{O}$.
$\prec, \preceq$	A preference relation.
$\prec_i, \preceq_i$	A preference relation on an attribute D_i.
$\prec_D$	The preference relation on the set of determined attributes.
$E(\prec)$	The size of the transitive closure of binary relation $\prec$.
$P(q)$	The set of objects partially dominating q

Table 1: The summary of frequently used notions.

Hotel name	Star level	Price
Amazing	5	180
Best View	4	160
Cindy's	3	165

Table 2: A set of hotels in Example 1.

In many applications, due to the large size of $\mathcal{O}$ or some other constraints, it is often infeasible to learn the user preference on $\mathcal{O}$ completely. Instead, the user preference can often be decomposed into preferences on attributes in $\mathcal{D}$ following the principles of preferences in [14].

A user preference $\prec_i$ on an attribute $D_i \in \mathcal{D}$ $(1 \leq i \leq d)$ is a strict partial order on the domain of D_i. That is, for two values $u, v \in D_i$, if $u \prec_i v$, the user prefers u than v. Again, we write $u \preceq_i v$ if $u \prec_i v$ or $u = v$.

The *assumption of independent attributes* is often made: a user's preference in space $\mathcal{D}$ is a composition of her/his preferences on all attributes. The *composition* of a set of preferences $\{\prec_1, \ldots, \prec_d\}$ on attributes $D_1, \ldots, D_d$ is a strict partial order $\prec = (\prec_1, \ldots, \prec_d)$ such that for any objects $o_1, o_2 \in \mathcal{O}$, $o_1 \prec o_2$ if and only if $o_1.D_i \preceq_i o_2.D_i$ for every attribute $D_i \in \mathcal{D}$ $(1 \leq i \leq d)$, and $o_1.D_{i_0} \prec_{i_0} o_2.D_{i_0}$ holds on at least one attribute $D_{i_0} \in \mathcal{D}$. As indicated in [14], such an assumption often holds in practice.

Example 1 (Preliminaries). Consider a customer's preference in choosing hotels. Suppose two factors matter: the price (the lower the better) and the star level (the higher the better). In the three hotels in Table 2, since Best View is more preferable than Cindy's in both star level and price, Best View $\prec$ Cincy's. Amazing is better than Best View in star level, but Best View is cheaper than Amazing. Thus, the customer does not prefer one than the other between Amazing and Best View. ∎

2.2 Problem Definition

On some attributes such as price, customers often have well accepted and consistent preferences. For example, all customers prefer low price. On some other attributes such as hotel brands, airlines, and locations, however, the preferences may vary dramatically from one user to another.

An attribute is called *determined* if a preference is defined on the attribute for all users. An attribute is called *undetermined* if there is no preference defined on the attribute for

Object	A	B	C
o_1	a_1	b_1	c_1
o_2	a_1	b_1	c_2
o_3	a_2	b_1	c_2
o_4	a_2	b_2	c_2
o_5	a_2	b_3	c_2

Table 3: The set of objects in Example 2.

all users. To keep our discussion simple, we focus on learning user preferences on undetermined categorical attributes.

In order to learn preferences on undetermined attributes, we need the input from a target user. In this study, we assume that a target user provides two types of input: a set S of superior examples and a set Q of inferior examples. An object o_1 is called a *superior example* if the target user specifies that, according to the customer's preference, o_1 is not dominated by any other objects o'. An object o_2 is an *inferior example* if the target user specifies that, according to the customer's preference, o_2 is dominated by some other objects o''.

In literature, a superior example is also called a *skyline point* or a *maximal vector* [16]. The skyline in a data set is the complete set of all skyline points. A point is inferior if it is not a skyline point. In a large data set, there can be many superior examples. Please note that, in our model, the target user is *not* required to specify *all* superior and inferior examples. Instead, a user may only provide a small number of superior and inferior examples. The data mining task is to learn the user's preferences as much as possible using those examples.

Without loss of generality, let $\mathcal{D} = \mathcal{D}_D \cup \mathcal{D}_U$ such that $\mathcal{D}_D \cap \mathcal{D}_U = \emptyset$, where $\mathcal{D}_D = \{D_1, \ldots, D_{d'}\}$ $(0 \leq d' < d)$ is the set of determined attributes and $\mathcal{D}_U = \{D_{d'+1}, \ldots, D_d\}$ is the set of undetermined attributes. Let $\prec_D$ be the preference defined on $\mathcal{D}_D$. Given a set of objects $\mathcal{O}$, a set $S \subseteq \mathcal{O}$ of superior examples and a set $Q \subset \mathcal{O}$ of inferior examples such that $S \cap Q = \emptyset$, a set of preferences $R = \{\prec_{d'+1}, \ldots, \prec_d\}$ is called a *satisfying preference set* (*SPS* for short) if (1) $\prec_{d'+i}$ $(1 \leq i \leq d - d')$ is a preference on attribute $D_{d'+i}$, and (2) according to the composite preference $\prec = (\prec_D, \prec_{d'+1}, \ldots, \prec_d)$, every object $o_1 \in S$ is not dominated by any other object $o' \in \mathcal{O}$, and every object $o_2 \in Q$ is dominated by at least one other object $o'' \in \mathcal{O}$.

Generally, given a set of superior and inferior examples, there may be no SPS, one SPS, or multiple SPSs.

Example 2 (SPS). Consider the set of objects in Table 3. Let A be a determined attribute where the preference is $a_1 \prec_A a_2$. Let B and C be two undetermined attributes.

Suppose a user specifies the set of superior examples $S_1 = \{o_1, o_3\}$ and the set of inferior examples $Q_1 = \{o_2\}$. The user does not label objects o_4 and o_5. Object o_2 is inferior, and o_1 is the only object that can dominate o_2 in the data set. Thus, examples o_1 and o_2 suggest $c_1 \prec_C c_2$ on attribute C. On the other hand, since o_3 is superior, if $c_1 \prec_C c_2$, then o_3 is dominated by o_1. In other words, examples o_1 and o_3 suggest $c_1 \not\prec_C c_2$. Therefore, there does not exist a SPS with respect to S_1 and Q_1.

Suppose another user specifies the set of superior examples $S_2 = \{o_1\}$ and the set of inferior examples $Q_2 = \{o_4\}$. It is easy to verify that both $\{b_1 \prec_B b_2, c_1 \prec_C c_2\}$ and $\{b_3 \prec_B b_2\}$ are SPSs with respect to S_2 and Q_2. In other words, the SPSs are not unique in this case. ∎

Suggested by Example 2, we study two problems in this paper. The first problem is whether a SPS exists.

Problem 1 (SPS existence). *Given a set of superior examples S and a set of inferior examples Q, determine whether there exists at least a SPS R with respect to S and Q.* ∎

As elaborated by the second case in Example 2, multiple SPSs may exist with respect to a set of superior examples and a set of inferior examples. Then, how can we evaluate the quality of the SPSs?

To avoid overfitting, we advocate the *minimal SPSs* which are the *simplest hypotheses* that fit the superior and inferior examples. A preference is a strict partial order which can be represented as a binary relation. The *complexity* of a strict partial order can be measured by the cardinality of the transitive closure of the binary relation. The intuition is that a stronger preference relation sets preferences between more pairs of objects than a weaker preference relation. Technically, let $\prec$ be a strict partial order. The *complexity* of $\prec$ is denoted by $|E(\prec)|$, where $E(\prec)$ denotes the transitive closure of $\prec$ as a binary relation.

The second problem about preference mining is to find a minimal SPS.

Problem 2 (Minimal SPS). *For a set of superior examples S and a set of inferior examples Q, find a SPS $R = \{\prec_{d'+1}, \ldots, \prec_d\}$ with respect to S and Q such that $|E(\prec_{d'+1}, \ldots, \prec_d)|$ is minimized. R is called a* minimal SPS. ∎

As described in Section 2.1, under the assumption of independent attributes, the preference in a multidimensional space is the composition of the preferences in all attributes. The complexity of the preference in a multidimensional space can be derived by the following rule.

Theorem 1 (Multidimensional preferences). *In space $\mathcal{D} = D_1 \times D_2 \times \cdots \times D_d$, let $\prec_i$ $(1 \leq i \leq d)$ be a preference on attribute D_i, and $\prec = (\prec_1, \ldots, \prec_d)$. Then,*

$$|E(\prec)| = \prod_{i=1}^{d}(|E(\prec_i)| + |D_i|) - \prod_{i=1}^{d}|D_i| \quad (1)$$

where $|D_i|$ is the number of distinct values in attribute D_i in the data set. ∎

2.3 Computational Complexity Analysis

In this section, we study the complexity of the SPS existence problem and the minimal SPS problem. When we consider the two problems with l undetermined attributes, we call them the l-d SPS existence problem and the l-d minimal SPS problem, respectively.

Lemma 1. *The 2-d SPS existence problem is NP-complete.*
Proof sketch. The SPS existence problem is in NP. We can prove that the 2-d SPS existence problem is NP-complete by polynomial time reducing the 3SAT problem [6]. ∎

Theorem 2. *The existence problem is NP-complete, even when there is only one undetermined attributes (i.e., $l = 1$).*
Proof sketch. We can prove that there is a polynomial time redution from the l-d SPS existence problem to the $(l+1)$-d SPS existence problem, and vice versa. Following Lemma 1, the SPS existence problem is NP-complete. ∎

Clearly, the minimal SPS problem is more difficult than the SPS existence problem. For a set of preferences that does not satisfy the given superior and inferior examples, we define its complexity (Theorem 1) as infinity. Since the minimal SPS problem is not in NP, applying Theorem 2, the following theorem immediately follows.

THEOREM 3. *The minimal SPS problem is NP-hard.* ∎

3. RELATED WORK

User preferences have been well recognized important in many applications. Kießling [14] introduced an expressive theoretical framework for preferences. The framework considers preferences in a multidimensional space. A set of preference constructors are given for both categorical and numerical domains. Our study follows the preference construction framework in [14].

A statistical model of user preferences is presented in [12, 11]. In the statistical model, the frequency of an item in a data set depends on two factors: the user's preference and the accessibility of the item. Moreover, a user's preference on an item can be further modeled as a function on the features of the item as well as the user profile which can be approximated by the user's behavior history data.

A framework of expressing and combining preferences is proposed in [1]. A user can assign a preference score to items and a model is applied to combine preferences. However, the model is predefined and is not learnt from data.

Different from [14, 12, 11, 1] which are dedicated to modeling user preferences, this study focuses on mining preferences from examples.

The problem of mining user preferences has been accessed by some previous studies from some angles different from this study. For example, Holland *et al.* [7] develop the data driven preference mining approach to find preferences in user session data in web logs. The central idea is that the more frequently an item appears, the more preferable the item is. However, the mining methodology in [7] may not be accurate since it does not consider the accessibility of data items which is important as indicated by [11].

Most recently, the problem of context-aware user preference mining is addressed in [8]. The major idea is that user preferences may be transient when the context (e.g., the topics in web browsing) changes. The data-driven approach [7] is extended accordingly.

[7, 8] do not use any explicit preference examples provided by users. To this extent, [7, 8] are analogous to the unsupervised learning approaches in classical machine learning [19]. In this study, we advocate to use superior and inferior examples in preference mining. Our method is analogous to the supervised learning approaches [19, 5].

There are also studies on supervised mining user preferences. Cohen *et al.* [20] develop an approach to order objects given the feedback that an object should be ranked higher than another. Joachims [10] uses a support vector machine algorithm to learn a ranking function of web documents utilizing user query logs of the search engine. Both studies focus on mining the order of objects according to user preferences. However, in this paper, instead of ordering objects, we are interested in learning the preferences in attributes which are the reason why objects should be ranked in such an order. We mine the preference set on each attribute underneath the preferences on object level.

Object-id	D_1	D_2	D_3	D_4	Label
o_1	1	5	a_3	b_3	
o_2	1	6	a_2	b_1	Inferior
o_3	1	6	a_2	b_3	
o_4	2	2	a_1	b_1	
o_5	2	5	a_2	b_2	Inferior
o_6	3	1	a_4	b_3	
o_7	3	4	a_2	b_2	Inferior
o_8	6	1	a_5	b_1	Inferior
o_9	6	1	a_5	b_3	
o_{10}	6	2	a_1	b_1	Inferior

Table 4: A set of objects as the running example.

Inferior	$P(q)$	Condition $C_q(p)$
o_2	o_1	$C_{o_2}(o_1) = (a_3 \prec_3 a_2) \wedge (b_3 \prec_4 b_1)$
	o_3	$C_{o_2}(o_3) = (b_3 \prec_4 b_1)$
o_5	o_1	$C_{o_5}(o_1) = (a_3 \prec_3 a_2) \wedge (b_3 \prec_4 b_2)$
	o_4	$C_{o_5}(o_4) = (a_1 \prec_3 a_2) \wedge (b_1 \prec_4 b_2)$
o_7	o_4	$C_{o_7}(o_4) = (a_1 \prec_3 a_2) \wedge (b_1 \prec_4 b_2)$
	o_6	$C_{o_7}(o_6) = (a_4 \prec_3 a_2) \wedge (b_3 \prec_4 b_2)$
o_8	o_6	$C_{o_8}(o_6) = (a_4 \prec_3 a_5) \wedge (b_3 \prec_4 b_1)$
	o_9	$C_{o_8}(o_9) = (b_3 \prec_4 b_1)$

Table 5: The conditions in the running example.

Our problem is also related to the classification problem. But the existing classification methods cannot be applied to the preference mining problem. In the traditional classification model, a set of training examples are labeled and the distributions of classes in the data space is learnt. The prediction is on the class of an unseen case. In this study, the superior and inferior examples are based on the dominance relation, and the relations between data points are learnt. The prediction is on, given two cases whose dominance relation is unknown, whether one case dominates the other.

User preferences are used in many applications, such as personalized recommendation systems [18] and preference queries on large databases [13, 3, 4, 15, 17].

4. A GREEDY METHOD

As indicated in Section 2.3, the SPS existence problem is NP-complete and the minimal SPS problem is NP-hard. Any polynomial time approximation algorithm cannot guarantee to find a SPS whenever a SPS exists. In other words, such an approximation algorithm may fail to find a SPS in some cases where a SPS does exist. Moreover, any polynomial time approximation algorithm cannot guarantee the minimality of the SPSs found.

In this section, we develop a greedy method to mine a simple SPS with respect to a set of superior examples S and a set of inferior examples Q. Our focus is on the practicality of the method – being simple and easily implementable.

For a set of preferences R on the undetermined attributes, an object $q \in Q$ is called *satisfied* if q is an inferior object with respect to composite preference $(\prec_D, R)$. Similarly, a point $s \in S$ is called *satisfied* if s is a superior object with respect to $(\prec_D, R)$. In Section 4.1, we give a method to satisfy the inferior examples in Q. In Section 4.2, we describe how to satisfy the superior examples in S.

4.1 Satisfying Inferior Objects

For two objects $o_1, o_2 \in \mathcal{O}$, consider the preference $\prec_D$ on

the determined attributes. If $o_1 \preceq_D o_2$, whether o_1 dominates o_2 depends on the preferences on the undetermined attributes. o_1 is said to *partially dominate* o_2. On the other hand, if $o_1 \not\preceq_D o_2$, then no preferences on the undetermined attributes can make $o_1 \prec o_2$ in space $\mathcal{D}$.

For each object $q \in Q$, let $P(q)$ be the set of objects in $\mathcal{O}$ that partially dominate q. If there exists an object $p \in P(q)$ such that $p \prec_D q$ and $p.D_U = q.D_U$, then q is satisfied. In such a case, q is said to be *trivially satisfied* by $\prec_D$. We can remove q from Q.

Suppose q is not trivially satisfied. For any set of preferences R on the undetermined attributes, if R satisfies q, there must exist at least one object $p \in P(q)$ such that $p \prec_R q$, where $\prec_R$ is the composite preference of R. We call $C_q(p) = p \prec_R q$ a *condition* of q being an inferior example.

Technically, $C_q(p) = \wedge_{d' < i \leq d, p.D_i \neq q.D_i}(p.D_i \prec_i q.D_i)$. Each conjunctive element $(p.D_i \prec_i q.D_i)$ is a *preference term* (or *term* for short) of preference $\prec_i$. A condition consists of preference terms on multiple undetermined attributes. And a preference term can appear in many conditions. A condition is satisfied if and only if all terms in the condition are satisfied.

In implementation, we build an in-memory R-tree on determined attributes to facilitate detecting whether an inferior object is trivially satisfied and computing the conditions of inferior objects.

Example 3 (Conditions). Consider the set of objects in Table 4 in space $\mathcal{D} = D_1 \times D_2 \times D_3 \times D_4$. Both D_1 and D_2 are numeric attributes where small values are preferred. D_3 and D_4 are undetermined. Suppose a user specifies a set $Q = \{o_2, o_5, o_7, o_8, o_{10}\}$ of inferior examples.

Since $o_4 \prec o_{10}$, o_{10} is trivially satisfied, and thus is removed. Table 5 shows the conditions of the other 4 inferior examples. ∎

Intuitively, in order to satisfy all inferior examples in Q, we have to satisfy at least one condition for each object in Q. We can select one condition for each inferior example, and then build a SPS by satisfying all selected conditions. The total solution space is of size $\prod_{q \in Q} |P(q)|$. An exhaustive method to find the minimal SPS enumerates all possible solutions and outputs one SPS with the minimum complexity. Clearly, an exhaustive method is computationally prohibitive in practice. To tackle the problem, we develop two greedy algorithms to find approximate minimal SPSs.

4.1.1 A Term-Based Algorithm

Let $R = (\prec_{d'+1}, \ldots, \prec_d)$ be the SPS to be computed. Initially, we set $\prec_i = \emptyset$ for each undetermined attribute D_i $(d' < i \leq d)$. We iteratively add a term t into a preference $\prec_i$ until all inferior examples in Q are satisfied. The utility of a term t on attribute D_i is measured by two factors: (1) *complexity increment* $CI(t)$ which is the increase of size of $E(R)$ if t is selected, and (2) *inferior example coverage* $Cov(t)$ which is the number of interior examples newly satisfied if t is selected.

To keep the complexity of $E(R)$ low, the smaller the complexity increment and the larger the inferior example coverage, the better a term. We define a utility score of a term t as $score(t) = \frac{Cov(t)}{CI(t)}$, and select the term with the largest utility score in each iteration. Algorithm 1 describes the algorithm framework.

Algorithm 1 The term-based greedy algorithm.

Input: the set of objects $\mathcal{O}$ and a set $Q \subset \mathcal{O}$ of inferior examples;
Output: a SPS $R = \{\prec_{d'+1}, \cdots, \prec_d\}$;
Description:
1: initialize all $\prec_i = \emptyset$ $(d' < i \leq d)$;
2: **for all** $q \in Q$ **do**
3: compute $P(q)$;
4: **if** q is not trivially satisfied **then**
5: **for all** $p \in P(q)$ **do**
6: compute condition $C_q(p)$
7: **end for**
8: **else**
9: $Q = Q \setminus \{q\}$
10: **end if**
11: **end for**
12: **while** $Q \neq \emptyset$ **do**
13: update $score(t)$ for each preference term t;
14: find the term t with the largest $score(t)$ value;
15: **if** t conflicts with previous selected terms **then**
16: remove t;
17: **else**
18: include t into R;
19: **for all** $q \in Q$ **do**
20: **for all** $p \in P(q)$ **do**
21: **if** $C_q(p)$ is satisfied **then**
22: $Q = Q \setminus \{q\}$; **break**;
23: **end if**
24: **end for**
25: **end for**
26: **end if**
27: **end while**
28: **return** R

Note that we cannot select a term which conflicts with the terms selected in previous iterations. That is, if a term $a_1 \prec a_2$ is already selected, then the conflicting term $a_2 \prec a_1$ cannot be selected, because a partial order is anti-symmetric. Such a conflicting term is removed in lines 15 and 16.

Computing Complexity Increment.

For a preference on an attribute, if a preference term is selected, multiple pairs may be added into the transitive closure of the updated preference.

Example 4 (Implied preference terms). Suppose on attribute D_3, we have $a_1 \prec_3 a_2$. If we add a new term $a_3 \prec_3 a_1$, due to the transitivity of partial orders, we have $a_3 \prec_3 a_2$ implied. Thus, the complexity increment on the attribute is 2. ∎

For a preference $\prec_i$ and a term t on attribute D_i $(d' < i \leq d)$, if a term t' holds in preference transitive closure $E(\prec_i \cup \{t\})$ but not in $E(\prec_i)$, then t' is called an *implied preference term* from t and $\prec_i$. $Imp(t, \prec_i)$ is the set of all implied terms from t and $\prec_i$. Trivially, $t \in Imp(t, \prec_i)$.

As mentioned in Section 2, a preference on an attribute can be represented as a directed acyclic graph. We maintain the transitive closure of $\prec_i$ on each undetermined attribute D_i. Once a term t on D_i is selected, we can compute $Imp(f, \prec_i)$ in time $O(|D_i|)$ where $|D_i|$ is the cardinality of D_i. Using Theorem 1, we can compute the complexity of the new preference and derive the complexity increment.

Example 5 (complexity increment). In Table 4, $R = \{\prec_3, \prec_4\}$. $|D_3| = 5$ and $|D_4| = 3$. Suppose, before an iteration, $\prec_3 = \{a_1 \prec_3 a_2\}$ and $\prec_4 = \emptyset$. Applying Theorem 1,

$|E(R)| = (1+5)\times 3 - 5\times 3 = 3$. If $a_3 \prec_3 a_1$ on D_3 is selected in the iteration, as shown in Example 4, $|E(\prec_3)| = 3$. So $|E(R)| = (3+5)\times 3 - 5\times 3 = 9$. Therefore, the complexity increment of $a_3 \prec_3 a_1$ is $CI(a_3 \prec_3 a_1) = 9 - 3 = 6$. Similarly, $CI(b_1 \prec_4 b_2) = 6$. ∎

Computing Inferior Example Coverage.

A condition $C_q(p)$ of an inferior example q consists of at most $(d - d')$ terms, each on one undetermined attribute. We write $t \in C_q(p)$ if t is a conjunctive element of $C_q(p)$. We also denote by $|C_q(p)|$ the number of terms in condition $C_q(p)$. $|C_q(p)| \leq d - d'$. If a term $t \in C_q(p)$ is selected, then $\frac{1}{|C_q(p)|}$ of terms in $C_q(p)$ are satisfied. Thus, we define the coverage of term t over condition $C_q(p)$ as

$$Cov(t, C_q(p)) = \begin{cases} \frac{1}{|C_q(p)|} & \text{if } t \in C_q(p) \\ 0 & \text{if } t \notin C_q(p) \end{cases}$$

Example 6 (Coverage of term over condition). In Table 5, the coverage of term $b_3 \prec_4 b_1$ over condition $C_{o_2}(o_1)$ is $Cov(b_3 \prec_4 b_1, C_{o_2}(o_1)) = \frac{1}{2}$. Similarly, $Cov(b_3 \prec_4 b_1, C_{o_2}(o_3)) = 1$, $Cov(b_3 \prec_4 b_1, C_{o_8}(o_6)) = \frac{1}{2}$, and $Cov(b_3 \prec_4 b_1, C_{o_8}(o_9)) = 1$. ∎

How should we define the coverage of a term over an inferior example? The above example indicates that a term t can appear in more than one condition of an inferior example, and it can appear in the conditions of many inferior examples. Moreover, we also have to consider the terms implied from t. Let us see the example below.

Example 7 (Coverage of implied terms). In Table 5, suppose we already have term $b_3 \prec_4 b_1$ on D_4. Then $b_1 \prec_4 b_2$ implies $b_3 \prec_4 b_2$. Because $Cov(b_1 \prec_4 b_2, C_{o_5}(o_4)) = \frac{1}{2}$ and $Cov(b_3 \prec_4 b_2, C_{o_5}(o_1)) = \frac{1}{2}$, if $b_1 \prec_4 b_2$ is selected, then $\frac{1}{2}$ of $C_{o_5}(o_4)$ and $\frac{1}{2}$ of $C_{o_5}(o_1)$ will be satisfied. Furthermore, $\frac{1}{2}$ of o_5 will be satisfied. ∎

To sum up the above discussion, if a term t and its implied terms appear in many conditions of an inferior example q, the coverage of t over q is the largest coverage of t and its implied terms over one condition of q. The reason is that we only need to satisfy one condition in order to satisfy an inferior example. Formally,

$$Cov(t, q) = \max_{t' \in Imp(t), p \in P(q)}\{Cov(t', C_q(p))\}.$$

Finally, the total inferior example coverage of t is the sum of the coverage of t over all inferior examples, that is,

$$Cov(t) = \sum_{q \in Q} Cov(t, q).$$

Example 8 (Inferior Example Coverage). Continue Example 6, we have $Cov(b_3 \prec_4 b_1) = \max\{0.5, 1\} + \max\{0.5, 1\} = 2$.

If $b_1 \prec_4 b_2$ is already selected before, then $b_3 \prec_4 b_1$ implies $b_3 \prec_4 b_2$, therefore $Cov(b_3 \prec_4 b_1)$ changes to 3. ∎

We elaborate the term-based greedy method in the following example.

Example 9 (The term-based greedy algorithm). We run the term-based greedy algorithm on our running example (Tables 4 and 5). Table 6 shows the utility score (Cov/CI) of each term in each iteration. Once a term is selected in an iteration (marked by *), the conditions of all satisfied inferior examples are removed, and some terms are also removed if they do not appear in any surviving condition (e.g., $a_4 \prec_3 a_5$ in iteration 1).

Term \ Iteration		1	2	3
D_3	$a_1 \prec_3 a_2$	1/3	1/4 *	\
	$a_3 \prec_3 a_2$	1/3	0.5/4	0.5/4
	$a_4 \prec_3 a_2$	0.5/3	0.5/4	0.5/4
	$a_4 \prec_3 a_5$	0.5/3	\	\
D_4	$b_1 \prec_3 b_2$	1/5	1/10	2/12 *
	$b_3 \prec_3 b_1$	2/5 *	\	\
	$b_3 \prec_3 b_2$	1/5	1/5	1/6

Table 6: A running example of term-based algorithm.

Algorithm 2 The condition-based greedy algorithm.

Description:

```
1-12: same as Algorithm 1
13: update score(C_q(p)) for each condition C_q(p);
14: find the condition C_q(p) with the largest score(C_q(p));
15: if any term of C_q(p) conflicts with previous selected
    terms then
16:     remove C_q(p);
17: else
18:     include all terms of C_q(p) into R;
19-28: same as Algorithm 1
```

After iteration 3, all inferior examples are satisfied. Finally, we obtain a SPS $R = \{\prec_3, \prec_4\}$ where $\prec_3 = \{a_1 \prec_3 a_2\}$ and $\prec_4 = \{b_1 \prec_4 b_2, b_3 \prec_4 b_1, b_3 \prec_4 b_2\}$. $|E(R)| = 21$. ∎

4.1.2 A Condition-Based Algorithm

The term-based algorithm selects one term in each iteration, eventually to satisfy at least one condition for each inferior example. Once the best term t is selected, the conditions containing t are likely to be satisfied very soon, since they have less terms left. However, if such a condition has a term t' with large complexity increment, then selecting t' will result in large complexity of the final result.

For example, in iteration 2 in Example 9, the best term is $a_1 \prec_3 a_2$. Once it is selected, $b_1 \prec_4 b_2$ is selected in the next iteration due to the fact that they are both in conditions $C_{o_5}(o_4)$ and $C_{o_7}(o_4)$. But the complexity increment of $b_1 \prec_4 b_2$ is large (12). Apparently, it may not be a good choice.

To overcome the deficiency of the term-based algorithm, we develop a condition-based algorithm which selects the best condition in each iteration, instead of the best term.

We define the inferior example coverage of a condition $C_q(p)$ to be the sum of the inferior example coverages of all its terms. That is,

$$Cov(C_q(p)) = \sum_{t \in C_q(p)} Cov(t).$$

We also apply Theorem 1 to compute the complexity increment of $C_q(p)$ by selecting all terms of $C_q(p)$. Please note that $CI(C_q(p))$ is not equal to the sum of the complexity increments of all its terms.

The utility of a condition is defined as $score(C_q(p)) = Cov(C_q(p))/CI(C_q(p))$. Algorithm 2 shows the condition-based algorithm modified from Algorithm 1. We only modify lines 13 to 18.

EXAMPLE 10 (THE CONDITION-BASED ALGORITHM). We run the condition-based greedy algorithm on the running example. Table 7 shows the utility score of each condition in each iteration. Once a condition is selected in an iteration (marked by *), the conditions of all satisfied inferior examples are removed.

All inferior examples are satisfied after iteration 3. We obtain a SPS with $\prec_3 = \{a_3 \prec_3 a_2, a_4 \prec_3 a_2\}$ and $\prec_4 = \{b_3 \prec_4 b_1, b_3 \prec_4 b_2\}$. $|E(R)| = 20$, which is smaller than that of the SPS obtained by the term-based algorithm. ∎

Inferior	Condition	1	2	3
o_2	$C_{o_2}(o_1)$	3/9	\	\
	$C_{o_2}(o_3)$	2/5 *	\	\
o_5	$C_{o_5}(o_1)$	2/9	1.5/10 *	\
	$C_{o_5}(o_4)$	2/9	2/16	\
o_7	$C_{o_7}(o_4)$	2/9	2/16	1/12
	$C_{o_7}(o_6)$	1.5/9	1.5/10	1/5 *
o_8	$C_{o_8}(o_6)$	2.5/9	\	\
	$C_{o_8}(o_9)$	2/5	\	\

Table 7: A running example of condition-based algorithm.

4.2 Satisfying Superior Objects

In the greedy algorithms described before, when a term (or a condition) is selected, the updated preferences may make some superior objects dominated by other objects. We call this term (condition) a *violating term (condition)*. A violating term (condition) cannot be selected and has to be removed from further consideration.

EXAMPLE 11 (VIOLATING CONDITION). Suppose o_3 is indicated as a superior example of objects in Table 4. In iteration 2 in Example 10, if $C_{o_5}(o_1)$ is selected, $a_3 \prec_3 a_2$ is selected. Thus, $o_1 \prec o_3$. $C_{o_5}(o_1)$ is a violating condition. ∎

We use superior objects as verifiers. In Algorithms 1 and 2, before we select the best term (condition) (line 18), we check whether it is a violating term (condition). If yes, it is removed.

5. EMPIRICAL STUDY

We conducted extensive experiments to study the effectiveness and the efficiency of our two greedy algorithms, using both synthetic data sets and real data sets. All algorithms were implemented in C++ and compiled by GCC. We ran experiments on a Pentium 4 2.8GHz PC with 512MB memory running Red Hat Linux operating system.

5.1 Synthetic Data Sets

A synthetic data set consists of d' determined attributes and $d - d'$ undetermined attributes. Each determined attribute is numerical. We use the benchmark data generator [2] to generate three types of distributions, *anti-correlated*, *independent*, and *correlated*. The partial order of an undetermined attribute D_0 is generated by combining two independent numerical attributes D_1' and D_2', that is, $D_0 = D_1' \times D_2'$. Then the domain size of D_0 is $|D_0| = |D_1'| \times |D_2'|$. For two values $(a_1, b_1), (a_2, b_2) \in D_0$ ($a_1, a_2 \in D_1', b_1, b_2 \in D_2'$), $(a_1, b_1) \prec_{D_0} (a_2, b_2)$ if (1) $a_1 \leq a_2$ and $b_1 \leq b_2$, and (2) $a_1 < a_2$ or $b_1 < b_2$. By default, a data set has 100,000 objects in a space with 3 determined attributes following independent distribution and 2 undetermined attributes. The domain size of an undetermined attribute is 50.

To choose superior and inferior examples, we first predefine the preference on every undetermined attribute. Then based on these preferences, we compute all superior objects and inferior objects. Finally, superior and inferior examples are randomly drawn from the set of superior and inferior objects, respectively. The default numbers of superior and inferior examples are both 40. Given a data set, the complexity of the minimal SPS and the running time vary a lot with respect to the selected examples, therefore in every experiment, we run our algorithms on 10 sets of randomly generated examples for the specified number of examples, and report the average SPS complexity and average running time.

We first compare the term-based greedy algorithm (TG) and the condition-based greedy algorithm (CG) with an exhaustive algorithm (EX). Then we study the effectiveness and efficiency of TG and CG.

5.1.1 Comparison with the Exhaustive Method

EX enumerates all possible solutions and outputs one SPS with the minimum complexity. The time complexity of EX is $O(n_p^{m_{inf}})$, where n_p is the average number of objects that partially dominate an object and m_{inf} is the number of inferior examples. We can only run EX on very small data sets with very few examples. Figure 1 shows the running time of TG, CG, and EX on data sets with 200 objects. The number of inferior examples is varied in the experiments, and it is equal to the number of superior examples. Running time is plotted in logarithmic scale. As expected, the running time of EX increases exponentially with respect to the number of examples, while TG and CG are much faster than EX.

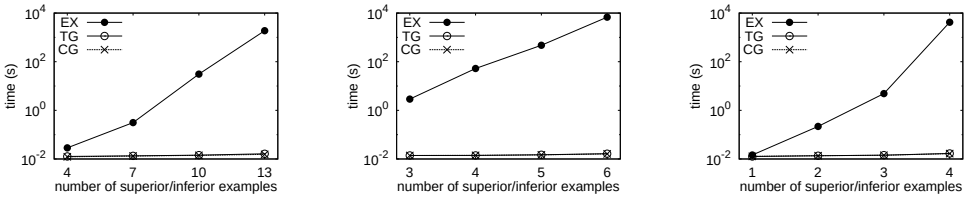
Objects in correlated data sets are much easier to be partially dominated than those in anti-correlated data sets. Hence, objects in correlated data sets have much more conditions than those in anti-correlated data sets, resulting in longer running time of EX on correlated data sets than that on anti-correlated data sets.

We use $ratio = \frac{complexity_G}{complexity_{EX}}$ to measure the effectiveness of our greedy algorithms. $complexity_{EX}$ is the complexity of the actual minimal SPS computed by EX, while $complexity_G$ is the approximate result obtained by TG or CG. Apparently, $ratio \geq 1$ and the smaller the value of $ratio$, the better the approximation. Figure 2 indicates that both TG and CG are very accurate ($ratio < 1.1$) on three types of data sets.

5.1.2 Effectiveness

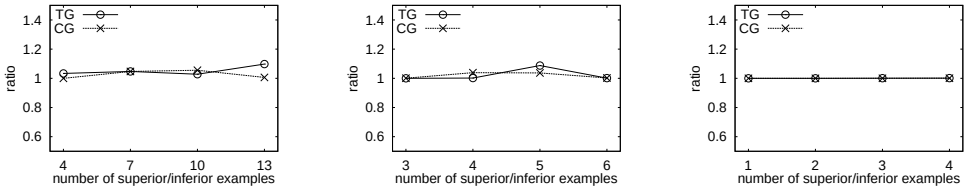
Since EX is too slow, we exclude it in the rest of experiments. To evaluate the effectiveness of our greedy algorithms, we use $ratio' = \frac{complexity}{cardinality}$ as the measurement. Here, $cardinality = \prod_{i=d'+1}^{d} |D_i|$ is the domain size of the space consisting of all undetermined attributes; while following Theorem 1, $complexity = \prod_{i=d'+1}^{d} |E(\prec_i)| - \prod_{i=d'+1}^{d} |D_i|$ is the complexity of the obtained SPS $\prec = (\prec_{d'+1}, \cdots, \prec_d)$. Intuitively, a smaller $ratio'$ represents a smaller SPS, hence, a better result.

We observe in Figure 3 that $ratio'$ on anti-correlated data sets are larger than that on independent data sets, and even larger than that on correlated data sets. Because in anti-correlated data sets, inferior examples are more likely to be dominated by different objects, thus have different condi-



(a) anti-correlated (b) independent (c) correlated

Figure 1: Running time compared with the exhaustive algorithm.



(a) anti-correlated (b) independent (c) correlated

Figure 2: Accuracy compared with the exhaustive algorithm.

tions and different preference terms. So more terms are selected. Moreover, the number of non-trivial examples (i.e., examples that are not trivially dominated) are few in correlated data sets, where the SPS complexity is even small.

Figure 3(a) shows that $ratio'$ rises when we increase the number of inferior examples from 20 to 100, since more preference terms are selected to satisfy more examples. However, the increment is sub-linear due to that some examples may share the same preference terms. Figure 3(b) shows that $ratio'$ is not sensitive to the number of superior examples. Because a superior example is used to eliminate wrong preference terms and its satisfaction does not increase the complexity. In Figure 3(c), we vary the domain size of undetermined attributes from 25 to 100. The complexity increases linearly.

Figure 3 also indicates that CG finds a SPS with smaller complexity than TG.

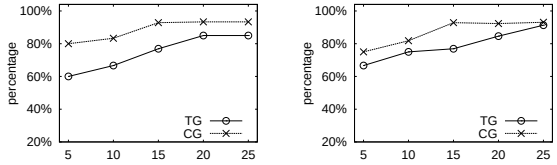
5.1.3 *Efficiency*

Figure 4 shows the running time of TG and CG on three types of data sets with the effects of different factors. Generally, both TG and CG run faster on correlated data sets while slower on anti-correlated data sets, because many examples in correlated data are trivially dominated.

Figure 4(a) shows the effect of increasing the size of data sets from $50,000$ to $200,000$. TG and CG has similar performance while CG is slightly faster than TG on anti-correlated data sets. The running time of both algorithms increases linearly.

Figures 4(b) and 4(c) vary the number of determined attributes from 2 to 5 and the number of undetermined attributes from 1 to 4, respectively. The running time increases linearly against the number of determined attributes while exponentially with respect to the number of undetermined attributes. When the number of undetermined attributes increases, the search space increases exponentially.

In Figures 4(d) and 4(e), we see that the running time rises linearly when the domain size of undetermined attributes and the number of inferior examples increase. Figure 4(f) shows that the number of superior examples has negligible effect on the efficiency of both algorithms.



(a) No. of inferior examples (b) No. of superior examples

Figure 5: Accuracy on NBA data set.

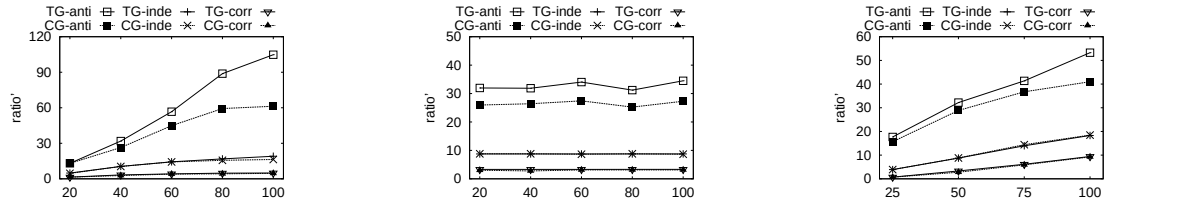
In conclusion, both two greedy algorithms are effective and practical. The condition-based algorithm is more accurate than the term-based algorithm and it is also faster on anti-correlated data sets.

5.2 Real Data Sets

We use the NBA data set (downloaded from `www.nba.com`) to evaluate the accuracy of our greedy algorithms. The data set contains the career average technical statistics of 3,924 players from 1946 to 2006. We select 5 attributes from the data set, the average points per game (PTS), the average steals (STL), the average blocks (BLK), the average rebounds (REB), and the average assists (AST). Large values are preferred on all attributes.

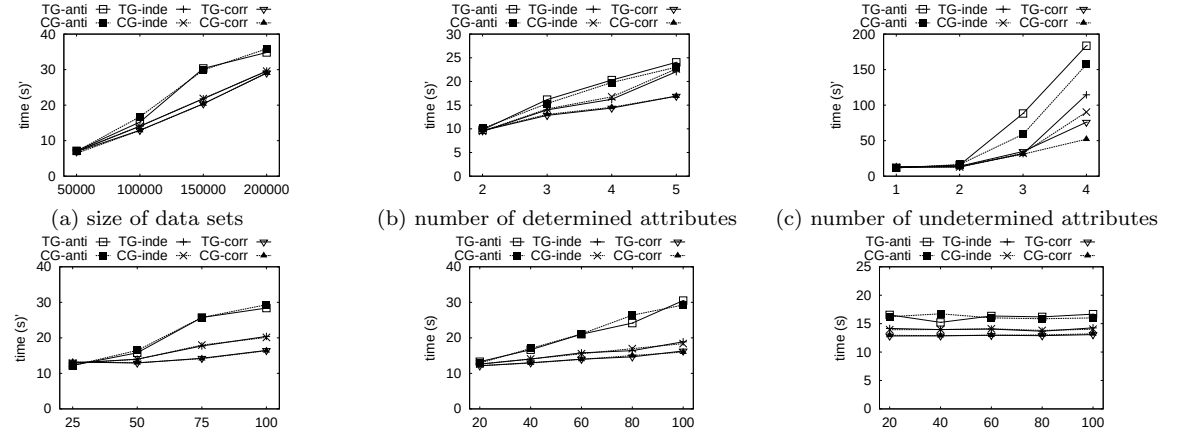
In this experiment, we use PTS, STL, and BLK as 3 determined attributes, and REB and AST as 2 undetermined attributes. In common sense, the larger values of REB and AST are prefered. The idea is that we hide the actual preferences on REB and AST. Then we use our greedy algorithm to mine the preferences with some inferior and superior examples. We want to evaluate whether the mined preferences are consistent with common sense. This is quantified by $pct = \frac{|R_{actual}|}{|R_{mined}|}$, where R_{total} is the set of mined preference terms and R_{actual} consists of terms in R_{total} which are consistent with common sense. Larger *pct* indicates higher accuracy.

We convert REB and AST into integers. REB has 23 distinct values while AST has 12. The inferior and superior examples are drawn from pre-computed inferior and superior objects using the actual preferences. Figure 5 shows *pct* with



(a) number of inferior examples (b) number of superior examples (c) domain size of undetermined attributes

Figure 3: $ratio'$**.**



(a) size of data sets (b) number of determined attributes (c) number of undetermined attributes

(d) domain size of undetermined attributes (e) number of inferior examples (f) number of superior examples

Figure 4: Running time.

different sets of inferior and superior examples. The running time is negligible.

In Figure 5(a), we use 15 superior examples. *pct* increases when the number of inferior examples increases from 5 to 25. Figure 5(b) shows that the result becomes more accurate when the number of superior examples increases from 5 to 25. The number of inferior examples is 15. *pct* is around 90% when we have more than 15 inferior and superior examples. Again, CG has better performance than TG. This experiment shows that in practice we often need a small number of inferior examples to learn user preferences accurately.

6. CONCLUSIONS

In this paper, we tackled a novel problem of mining user preferences using superior and inferior examples. We elaborated the applications of the problem and modeled the problem systematically. We showed that both the SPS existence problem and the minimal SPS problem are challenging. As the first attempt to tackle the problem, we devised a greedy method. The empirical study using both real data and synthetic data indicated that our greedy method is practical.

7. REFERENCES

[1] R. Aggarwal and E. Wimmers. A framework for expressing and combining preferences. In *SIGMOD*, 2000.

[2] S. Borzsonyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[3] J. Chomicki. Querying with intrinsic preferences. In *EDBT*, 2002.

[4] J. Chomicki. Database querying under changing preferences. *Annals of Mathematics and Artificial Intelligence*, 2006.

[5] R. Duda, P. Hart, and D. Stork. *Pattern Classification (2nd edition)*. John Wiley & Sons, New York, 2001.

[6] M. Garey and D. Johnson. *Computers and Intractability: a Guide to The Theory of NP-Completeness*. Freeman and Company, New York, 1979.

[7] S. Holland, M. Ester, and W. Kießling. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*, 2003.

[8] E. Jembere, M. O. Adigun, and S. S. Xulu. Mining context-based user preferences for m-services applications. In *WI*, 2007.

[9] B. Jiang, J. Pei, X. Lin, D. W-L Cheung, and J. Han. Mining preferences from superior and inferior examples. Technical report TR 2008-09, School of Computing Science, Simon Fraser University, 2008.

[10] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

[11] S. Y. Jung, J.-H. Hong, and T.-S. Kim. A formal model for user preference. In *ICDM*, 2002.

[12] S. Y. Jung, J.-H. Hong, and T.-S. Kim. A statistical model for user preference. *TKDE*, 2005.

[13] K. Govindarajan and B. Jayaraman and S. Mantha. Preference Queries in Deductive Databases. *New Generation Computing*, 2001.

[14] W. Kießling. Foundations of preferences in database systems. In *VLDB*, 2002.

[15] W. Kießling and G. Kostler. Preference SQL – design, implementation, experience. In *VLDB*, 2002.

[16] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 1975.

[17] M. Lacroix and P. Lavency. Preferences; putting more knowledge into queries. In *VLDB*, 1987.

[18] R. D. Lawrence, G. S. Almasi, V. Kotlyar, M. S. Viveros, and S. S. Duri. Personalization of supermarket product recommendations. *Data Min. Knowl. Discov.*, 2001.

[19] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[20] R. E. S. William W. Cohen and Y. Singer. Learning to order things. *J. Artif. Intell. Res. (JAIR)*, 1999.