Learning Conditional Preference Networks From Inconsistent Examples

Juntao Liu, Yi Xiong, Caihua Wu, Zhijun Yao, and Wenyu Liu, *Member, IEEE*

***Abstract*— The problem of learning Conditional Preference Networks(CP-nets) from a set of examples has received great attention recently. However, because of the randomicity of the users' behaviors and the observation errors, there is always some noise making the examples inconsistent, namely, there exists at least one outcome preferred over itself (by transferring) in examples. Existing CP-nets learning methods can not handle inconsistent examples. In this work, we introduce the model of learning consistent CP-nets from inconsistent examples and present a method to solve this model. We do not learn the CP-nets directly. Instead, we first learn a preference graph from the inconsistent examples, because dominance testing and consistency testing in preference graphs are easier than those in CP-nets. The problem of learning preference graphs is translated into a 0-1 programming and is solved by the branch-and-bound search. Then the obtained preference graph is transformed into a CP-net equivalently , which can entail a subset of examples with maximal sum of weight. Examples are given to show that our method can obtain consistent CP-nets over both binary and multivalued variables from inconsistent examples. The proposed method is verified on both simulated data and real data, and it is also compared with existing methods.**

***Index Terms*— preference learning, preference elicitation, conditional preference networks, preference graph, branch-and-bound.**

I. Introduction

Preference learning or preference elicitation is an important issue in many scientific fields such as decision theory, operations research, economics and computer science. Conditional Preference Networks(CP-nets) [1], as a simple and intuitive graphical tool for representing and reasoning users' conditional preference over outcome space, have received great attention in recent years. In the outcome space, each outcome is represented by the values of a set of variables. A CP-net is a digraph, whose nodes correspond to variables. Each node is annotated with a conditional preference table. The preferential dependency is represented by the graph structure. CP-nets have various extensions [2]–[8]. But because of the simple graphical structure, learning CP-nets in complex domains is a difficult task.

Juntao Liu is with the Department of Electrical and Information Engineering, Huazhong University of Science and Technology, Wuhan, P.R. China, 430074 and the Department of Computer Engineering, Ordnance Engineering College, Shijiazhuang, P.R China, 050003(E-mail: prolay@163.com).

Yi Xiong, Zhijun Yao and Wenyu Liu are with the Department of Electrical and Information Engineering, Huazhong University of Science and Technology, Wuhan, P.R. China, 430074(E-mail: bearone@live.cn, alexyau78@gmail.com, liuwy@mail.hust.edu.cn).

Caihua Wu is with the Department of Information Counterwork, Air Force Radar Academy, Wuhan, P.R.China, 430010 (E-mail: wucaihua_1999@yahoo.com.cn).

CP-nets learning methods can be divided into active learning [9], [10] and passive learning [11]–[13]. Different from active learners, passive learners do not interact with users. They collect the set $P = \{o_1 \succ o_1', o_2 \succ o_2', ..., o_m \succ o_m'\}$ of examples and weight $W = (w_1, w_2, ..., w_m)$ by observing the users passively, where an example $o_i \succ o_i'$ means "outcome o_i is strictly preferred over outcome o_i'", and the weight w_i denotes the importance or confidence of the ith example. Such set of examples may be gathered, for instance, by observing online user's choices. If a certain choice is observed more frequently, it must be more important and confident than others, and a higher weight should be assigned to the corresponding example. Because of randomicity of the users' behaviors and the observation errors, there is always some noise making the examples inconsistent, namely, there exists at least one outcome preferred over itself (by transferring) in P. It is impossible to derive a consistent CP-net entailing all examples. Here, a CP-net N is consistent means there are no outcomes preferred over themselves in N. An inconsistent CP-net can not represent a user's preference, for nobody prefers one thing over itself. So a CP-net learner should derive a consistent CP-net even if the learning examples are inconsistent. The derived CP-net only entails a subset of the inconsistent examples. To represent the user's preference accurately, the subset of examples entailed by the derived CP-net should be most important and confident, which requires the sum of weight of entailed examples to be maximized. In one word, the task of learning CP-nets from inconsistent examples is to derive a consistent CP-net entailing a subset of the examples with maximal sum of weight. Formally, we can give the model:

$$\begin{cases} \text{maximize: } \sum_{o_i \succ o_i' \in P_N} w_i \\ \text{subject to: CP-net } N \text{ is consistent} \end{cases} \tag{1}$$

where, P_N is the subset of the examples P entailed by CP-net N. It has been proved that deciding whether there exists a CP-net entailing all the examples is NP-hard [11]. So the formula (1) is also NP-hard. Existing methods [9]–[13], assuming the set P of examples is consistent, are different from our model and cannot solve formula (1). For instance, active learning method proposed in [9], [10] can not solve passive learning problem. The learning method proposed in [13] assumes every variable is preferentially independent. It means there are no edges in CP-nets. The algorithm proposed in [12] learns a CP-nets over a fixed acyclic digraph. All of the methods assume the learning examples are consistent and the constraint on structure of CP-nets restricts their application. In [11], conditional preference tables are learned by solving a set of SAT-2 problems. It is assumed that the learning examples are

Digital Object Indentifier 10.1109/TKDE.2012.231

transparently entailed by CP-nets (see Definition 5 in [11]). If the assumption is not satisfied, some of the SAT-2 problems are unresolvable.

To solve formula (1), one should decide whether a given CP-net N is consistent, which is required by the constraint. This process is called consistency testing. Dominance testing is also needed for one should decide whether an example is entailed by a given CP-net. Here, dominance testing means given a CP-net N and two outcomes o and o', deciding whether o is preferred over o' in N. Generally, dominance testing and consistency testing in CP-nets are difficult [14], [15]. In [16], preference graphs are used to represent the preference relations induced directly from CP-nets. A preference graph is able to represent a preference relation entailed by a CP-net equivalently. Further more, we discover dominance testing and consistency testing in preference graphs are easier than those in CP-nets. For these reasons, we do not learn CP-nets directly. Instead, we solve formula (1) by two steps: 1) Derive a consistent preference graph entailing a subset of the examples with maximal sum of weight; 2) Transform the obtained preference graph into a CP-net equivalently. We prove the obtained CP-net entails the subset of examples with maximal sum of weight.

The main contributions of our work are two-fold. 1) We propose the model of learning CP-nets from inconsistent examples. Although the problem of learning CP-nets has received great attention, the problem of learning CP-nets from inconsistent examples has seldom been addressed. 2) We propose a two-step method to solve this problem. Our method can learn general CP-nets over multivalued variables from inconsistent examples.

The rest of the paper is organized as follows. In section 2, the related work is introduced. In Section 3, we introduce necessary background on CP-net and Preference Graph. In Section 4, the methods of dominance testing and consistency testing in preference graphs are presented. Then our CP-nets learning method and its properties are given. Two examples are given to illustrate our method at the end of this section. In section 5, the proposed method is verified on simulated data and real data, and it's also compared with the method proposed in [11] and the methods for learning to rank. Finally, we conclude and mention some possible future researches.

II. Related Work

CP-net represents a partial order over outcome space, so learning CP-net can be regarded as learning an preference order from pairwise comparisons. This problem is also called learning to order things [20] or supervised ordering [21]. In this scenario, the learner is provided with a set of outcomes and a set of pairwise comparisons over these outcomes, and the task is to predict the preference order for a new set of outcomes. If the predicted preference order is required to be a total order, the problem is also called the ranking problem. To solve the problem of learning preference order, three general kinds of approaches have been proposed in the literature. The first one is based on utility function or value function [21]–[25]. The basic idea is to evaluate individual outcome by utility function. The second one is based on related preference order [20], in which binary relation between outcomes is learned. The third one is based on special preference model, such as Lexicographic Order Model [26], [27] and CP-net [9]–[13].

Approaches based on utility function assign a degree of preference to each outcome in the outcome space. The value of utility function can be numerical or ordinal. Learning such utility function becomes a regression problem. Some special constraints should be taken into consideration such as the bound of the bounded range and the monotonicity of the utility function. The utility function can represent a total order, while it cannot represent partial order. Kamishima et al. proposed expected rank regression method [21]. The training data is assumed to be generated from an unseen complete order randomly. The expected rank for each outcome is computed under this assumption. Then the utility function is learned by common regression method. The order is estimated based on Thurstonian model according to the utility function. Metzler and Croft [39] proposed a linear utility function and minimized the ranking error by coordinate ascent. RankBoost [22] method is proposed to solve the collaborative filtering where a user's preference is estimated from the other users' preferences. It can also be applied to supervised ordering. RankBoost, taking feedback function and ranking feature as input, is a boosting algorithm. For each round, the algorithm trains a weak learner and selects its weight, and updates the distribution of the examples. The final order is predicted according to the weighted sum of weak learner, which can be regarded as a utility function. The algorithm minimizes Kendall distance between learning examples and the predicted order. AdaRank [38] is also a boosting algorithm, and optimizes information retrieval performance measure function, such as Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain(NDCG), directly. The approaches based on SVM [23]–[25] estimate utility function using SVM. Order SVM [23] learns utility function to discriminate whether or not a given outcome is ranked higher then i. While, Support Vector Ordinal Regression [24] and Ranking SVM [25] estimate the utility function by maximizing the margin between preferred outcomes and non-preferred outcomes, and their formulations are very similar to standard SVM. Multi-Hyperplane Ranker [30]overcomes the shortcoming of Ranking SVM that a single hyperplane can't tackle complex ranking problems by training several base rankers using Ranking SVM. The orders got by base rankers are aggregated as the final order. Carvalho et al. [32] modified the objective function in Ranking SVM and introduced a sigmoid-based loss function to measure the ranking error, which can handle outliers in training examples robustly.

RankNet [28] optimizes ranking loss function represented by cross entropy cost function using neural network and obtains a ranking function. FRank [29] uses Fidelity loss function to replace the loss function in RankNet. LambdaRank [35] optimizes Normalized Discounted Cumulative Gain(NDCG) directly by simply multiplying gradient of loss function in RankNet by the size of the change in NDCG. LambdaMART [36] uses gradient in LambdaRank during the training procedure of Multiple Additive Regression Trees (MART) [37],

and can optimize NDCG directly. Ranking methods stemming from RankNet, such as LambdaRank, LambdaMART, gain great success in information retrieval. For an example, the ranking system aggregating LambdaRank and LambdaMART won Track 1 of the 2010 Yahoo! Learning To Rank Challenge [34]. In recent years, some list-wise ranking methods, which take order lists of items as training examples, are proposed, such as ListNet [31] and Top-k ListNet [33]. List-wise approaches are out of the scope of this paper.

Approaches based on related preference order typically learn a binary preference relation over outcomes, and the predicted preference order is extracted from binary preference relation. Because the learning examples can be used directly, these approaches are simpler than approaches based on utility function. But these approaches are generally required to minimize the number of outcome pairs conflicted with the learning examples, and this requirement leads to NP-hard problem. Some approaches can get good approximation. In Cohen's method [20] binary preference relation $\succ$ is learned to maximize the sum of the binary preference function $\sum_{o' \succ o''} PREF(o', o'')$ for all possible outcome pairs o', o''. This maximization problem is known as the weighted feedback arc set problem, and is NP-hard. It is not tractable to find the optimal solution if the size of outcome space is large. Hence, a greedy algorithm is proposed in [20], which can find approximate solution in $O(n^2)$. The other difficulty of these approaches is that the extracted preference order is normally not unique, because the preference relation may not consistent or complete.

Besides the approaches based on utility function and preference relation, there are approaches based on special preference models. These approaches assume the preferences are represented by special models. An example is the Lexicographic Order Model(LPM) [26], which defines important order over variables and uses this order to determine the preference order between outcomes. Schmitt and Martignon [26], proposed a greedy variable permutation algorithm guaranteeing to find one of the LPMs that is consistent with the learning examples, if one exists. They also proved that for the noisy learning examples, the problem of finding a LPM that does not violate more than a constant number of pairwise comparisons in learning examples is NP-complete. Based on this greedy algorithm Yaman et al. [27] proposed a method to predict the preference order from a set of LPMs consistent with the learning examples. The assumption of LPM is strong, for the preference on individual variable is sometimes independent on other variables.

CP-net is proposed to address the conditional independency between preferences on variables as mentioned earlier. As far as we know, there does not existstudy in learning CP-nets from inconsistent examples. In term of learning CP-nets from consistent examples, Koriche and Zanuttini [9], [10] proposed an active learning algorithm. Learnability of CP-nets is addressed in [11] and [17]. The results **show** the problem of learning CP-nets is intractable even under some simplifying assumptions. In [12] and [13], the problem of learning separable ceteris paribus structures (SCP-structures) is addressed. SCP-structures are the simplified CP-nets, in which every variable is preferentially independent on all other variables. The algorithm for learning CP-nets over a fixed acyclic digraph is proposed in [12]. In [11], an algorithm for generating an acyclic CP-net entailing all examples is proposed. It is proved that the algorithm is a PAC-learner if the learning examples are transparently entailed by CP-nets (see Definition 5 in [11]).

In this paper, we propose an approach for learning CP-nets from inconsistent pair-wise examples. The main differences between previous approaches and ours are: 1) Compared with utility function based approaches and related preference order based approaches, our approach learns a preference model, CP-net, rather than the utility function (order function) or related preference order, which is difficult to represent conditional preference. 2) Compared with other CP-net learning approaches, our approach can learn CP-nets from inconsistent training examples. We don't require that all of the training examples are entailed by CP-nets. Our method can be applied widely than other CP-nets learning methods. 3) Compared with active CP-net learning approaches, our approach learns CP-net passively. The training examples are collected silently and the users don't need to be disturbed.

III. CP-nets and Preference Graphs

We start by briefly reviewing the basics of CP-net. Let $V = \{X_1, X_2, ..., X_n\}$ be a set of variables. Each variable X_i is associated with a domain of value $D(X_i)$, which is a finite set. In this paper, we do not restrict variables to be binary. For set $U \subseteq V$ of variables, $D(U) = \prod_{X_i \in U} D(X_i)$. $\Omega = \prod_{X_i \in V} D(X_i)$ denotes the outcome space. $o \in \Omega$ is called an outcome. $o[U]$ denotes the projection of o on set $U \subseteq V$ of variables.

Preference relation $\succ$ is the strict partial order on Ω, i.e. an irreflexive, transitive and asymmetric relation. $o \succ o'$ means "outcome o is strictly preferred over outcome o'".

A conditional preference rule (CPR) on a variable $X \in V$ is an expression: $u : x \succ x'$, where $x, x' \in D(X)$, u is the assignment of set $U \subseteq V/X$ of variables, and the variables in U are called the parent variables of X denoted by $Pa(X)$. Such a rule means "given that u hold, x is preferred over x', all the other variables being equal". A conditional preference table (CPT) on a variable X, $CPT(X)$, is the set of CPRs on X. If for any two values x and x' of variable X, a CPR is associated with each assignment of $Pa(X)$, $CPT(X)$ is complete.

Definition 1 [16]. A **CP-net** N over set $V = \{X_1, X_2, ..., X_n\}$ of variables is a labeled digraph over V, each node X_i is annotated with a conditional preference table $CPT(X_i)$, with respect to the set $Pa(X_i)$ of parents of X_i in the digraph. If conditional preference tables on all variables in V are complete, the CP-net is **complete**.

In [16], Preference Graphs are used to describe the preference relation between outcomes directly induced from CP-nets. But the preference graphs were not defined in [16] formally. Now we give the formal definition of preference graphs.

Definition 2. A **Preference graph** G is a digraph over outcome space Ω. Each node is corresponding to an outcome

in Ω. For any two outcomes $x, y \in \Omega$ differing on only one variable, if $x \succ y$, $(y, x) \in E$, where E is the set of edges in G. If for any two outcomes x and y differing on only one variable, $(y, x) \in E$ or $(x, y) \in E$ hold, the preference graph is **complete**.

Note there are no direct edges in preference graphs between any two outcomes differing on more than one variables. So each node in the preference graph connects with $m = \sum_{i=1}^{n} (|D(X_i)| - 1)$ other nodes at most. There are $k = m|\Omega|/2$ edges in a preference graph at most. Here, $|S|$ denotes the number of elements in set S.

Let $\succ_N$ denote the preference relation entailed by a CP-net N. The CP-net N is consistent iff for any $o \in \Omega$, $o \not\succ_N o$ hold. Similarly, let $\succ_G$ denote the preference relation entailed by a preference graph G. The preference graph G is consistent iff for any $o \in \Omega$, $o \not\succ_G o$ hold. Because a CP-net can be induced to a preference graph directly, deciding whether a CP-net is consistent can be transformed into deciding whether the induced preference graph is consistent. For a preference graph G, if $x \succ_G y$, there exists a path from node y to node x in preference graph G [16]. The sequence of nodes on this path is called an improving flipping sequence [16]. In an acyclic preference graph, there is no directed path from one node back to itself, namely, there is no outcome preferred over itself. So acyclic preference graph is consistent. Deciding whether a preference graph is consistent can be transformed into deciding whether the preference graph is acyclic.

IV. Learning CP-nets from Inconsistent Examples

Definition 3. Let $P = \{o_1 \succ o'_1, o_2 \succ o'_2, ..., o_m \succ o'_m\}$ denote the set of examples, if there exists an outcome o preferred over itself (by transferring) in P, the set P of examples is **inconsistent**.

To learn a CP-net from inconsistent examples is to derive a consistent CP-net entailing a subset of the examples with maximal sum of weight. The model is shown in formula (1). We solve this model by two steps: 1) Derive a consistent preference graph entailing a subset of the examples with maximal sum of weight; 2) Transform the obtained preference graph into CP-net equivalently. Note we do not restrict variables in V to be binary. The proposed method can handle both binary and no-binary variables.

This section describes the proposed method of learning CP-nets from inconsistent examples. Firstly, we present the methods of dominance testing and consistency testing in preference graphs. Secondly, we present the method of learning preference graphs from inconsistent examples. We translate this learning problem into a 0-1 programming and solve it by branch-and-bound search. Thirdly, we present the algorithm for transforming preference graphs into CP-nets equivalently. Every pair of edges in a preference graph is analyzed to find the preferential dependency between variables, and then conditional preference tables on all variables are generated. Fourthly, we present the properties of the method. We prove the obtained CP-net entails the subset of examples with maximal sum of weight. Finally, we give two examples to illustrate our method.

A. Learning Preference Graphs

To solve the model in formula (1), dominance testing and consistency testing are needed. Here, dominance testing means given two outcomes o and o', deciding whether $o \succ o'$. Consistency testing means deciding whether a given CP-net N or preference graph G is consistent. The following Theorem and Corollary give dominance testing and consistency testing methods in preference graphs represented by adjacency matrices.

Theorem 1 (Dominance Testing). Let A be the adjacency matrix of a preference graph G. $o_j \succ_G o_i$, iff $(e^A - I)_{i,j} \neq 0$, where $(M)_{i,j}$ denotes the element of matrix M at the ith row and jth column.

Proof: A^k is the connection of preference graph G after k jumps, and $(A^k)_{i,j}$ is the number of paths from node i to node j after k jumps. While $e^A - I = \sum_{k=1}^{+\infty} A^k/k!$, iff $(e^A - I)_{i,j} \neq 0$, there is at least one path from node i to node j, namely, $o_j \succ_G o_i$. ■

By theorem 1, We only need compute $e^A - I$ to get the preference relation between any two outcomes. Furthermore, we can use theorem 1 to decide whether a given preference graph is consistent.

Corollary 1 (Consistency Testing). Let A be the adjacency matrix of a preference graph G. G is consistent, iff $(e^A - I)_{i,i} = 0$ for all i.

Generally, dominance testing and consistency testing in CP-nets are difficult. Both dominance testing and consistency testing are PSPACE-complete [14], [15]. By theorem 1 and corollary 1, we only need compute $e^A - I$ for dominance testing and consistency testing in preference graphs. While, $e^A - I$ can be computed in polynomial time [18]. So dominance testing and consistency testing in preference graphs are easier than those in CP-nets.

Definition 4. Given the set $P = \{o_1 \succ o'_1, o_2 \succ o'_2, ..., o_m \succ o'_m\}$ of examples and their weight $W = (w_1, w_2, ..., w_m)$. Let $P_G = \{o \succ o' | o \succ_G o', o \succ o' \in P\}$ be the subset of P entailed by a preference graph G. Let $G* = \arg\max_G \sum_{(o_i \succ o'_i) \in P_G} w_i$. P_{G*} is called the maximal satisfiable subset of examples P.

To learn a preference graph from inconsistent examples is to find the maximal satisfiable subset of examples, and find a preference graph entailing them. For the set $P = \{o_1 \succ o'_1, o_2 \succ o'_2, ..., o_m \succ o'_m\}$ of examples and weight $W = (w_1, w_2, ..., w_m)$, we construct the training matrix A_P. If $o_j \succ o_i$ is the kth examples in P, $(A_P)_{i,j} = w_k$; otherwise, $(A_P)_{i,j} = 0$. Where, i and j are the outcome indices in outcome space. Let matrix A be the adjacency matrix of preference graph G. By theorem 1, $th(e^A - I)$ is the preference relation represented by G, where $th(M)$ is the threshold function. If $M_{i,j} > 0$, $(th(M))_{i,j} = 1$; otherwise, $(th(M))_{i,j} = 0$. The sum of weight of entailed examples is $g(A) = tr(A_P^T th(e^A - I))$, where $tr(M)$ is the trace of matrix M. From formula 1, the problem of learning preference graphs from inconsistent examples can be described as following:

$$\text{maximize}: \quad g(A) = tr(A_P^T th(e^A - I)) \tag{2}$$

$$\text{subject to}: \quad (e^A - I)_{i,i} = 0 \tag{3}$$

$$(A)_{i,j} = 0, 1 \quad (4)$$
$$(A)_{i,j} + (A)_{j,i} \leq 1 \quad (5)$$
$$\sum_i (A)_{i,j} \leq m \quad (6)$$
$$\sum_j (A)_{i,j} \leq m \quad (7)$$
$$m = \sum_{i=1}^{n} (|D(X_i)| - 1) \quad (8)$$
$$\sum_i \sum_j (A)_{i,j} \leq k \quad (9)$$
$$k = \frac{1}{2} m \prod_i |D(X_i)| \quad (10)$$

Where A is the adjacency matrix of the target preference graph; A_P is the training matrix; Formula (3) means the preference graph should be consistent by corollary 1. Formula (4) means the adjacent matrix A is a 0-1 matrix. Formula (5) means there is one preference relation between any two outcomes at most. Formula (6)-(8) mean each node in the preference graph connects with m other nodes at most. Formula (9)-(10) mean the number of edges in a preference graph cannot exceed k.

B. Branch-and-bound Search

The above problem is a nonlinear 0-1 programming. We can solve this problem through branch-and-bound search. In fact, according to Definition 2, a node in preference graph can only connect with the nodes that differ on only one variable from it. Let $\{e_1, e'_1, e_2, e'_2, ..., e_k, e'_k\}$ be the possible edges in preference graph G, where k is defined in formula (10), e_i and e'_i are the edges between same nodes with opposite directions. e_i and e'_i cannot exist in a consistent preference graph simultaneously. We construct a k-level search tree, as shown in Fig.1, in which each node is associated with a vector $B = (b_1, b_2, ..., b_k)$. $b_i = 1$ means $e_i \in E, e'_i \notin E$, where E denotes the set of edges in the target preference graph; $b_i = 0$ means $e_i \notin E, e'_i \in E$; $b_i = -1$ means $e_i \notin E, e'_i \notin E$; For convenience, let $b_i = \infty$ mean $e_i \in E, e'_i \in E$, although the represented preference graph is inconsistent. Let $A(B)$ denote the adjacency matrix of the preference graph represented by vector B. Root node is associated with $(-1, ..., -1)$. For a node v in search tree at level h, the associated vector is $B = (b_1, b_2, ...b_h, -1, ..., -1)$ with $b_1, b_2, ...b_h = 0$ or 1, and the node v has two children associated with $(b_1, b_2, ...b_h, 0, -1, ..., -1)$ and $(b_1, b_2, ...b_h, 1, -1, ..., -1)$, respectively. Every node in the search tree is corresponding to a preference graph represented by the associated vector, while every leaf node is corresponding to a complete preference graph. In order to find the maximum value of function $g(A)$ in formula (2), we can search the tree by branch-and-bound search.

Theorem 2. Let G and G' be the preference graphs over the same outcome space. E and E' are the sets of edges in G and G', respectively. And $E \subseteq E'$. If G is inconsistent, then G' is also inconsistent.

Proof: Because G is inconsistent, there exists an outcome o preferred over itself, then there is a path from node o to node o in E. Because $E \subseteq E'$, this path exists in E' too. G' is also inconsistent. ■

Let v' be a node in the subtree rooted at v. The corresponding preference graphs of node v and node v' are G and G', respectively. The sets of edges in them are E and E', respectively. Because node v' is a descendent of node v, we have $E \subseteq E'$. By theorem 2, if G is inconsistent, G' is inconsistent too. We can prune the subtree rooted at v, if the corresponding preference graph of node v is inconsistent.

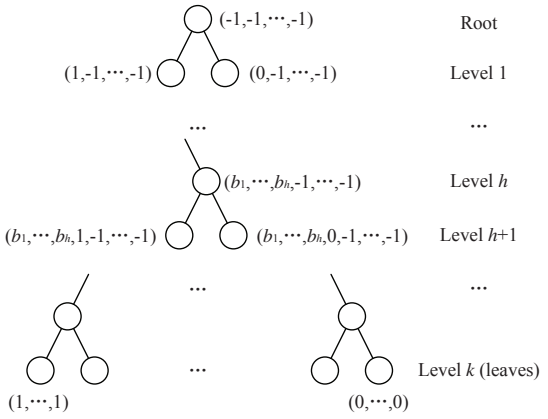

Fig. 1. Search Tree and Associated Vectors.

The key to branch-and-bound search is to provide upper and lower bound of function $g(A)$ in formula (2) at every node in the search tree. So we first give the monotonicity of function $g(A)$, and then give the upper and lower bound of it.

Theorem 3. Let E and E' be the sets of edges in the preference graphs G and G', respectively. The adjacency matrixes of G and G' are A and A' , respectively. If $E \subseteq E'$, $g(A) \leq g(A')$.

Proof: The function $g(A)$ in formula (2) is the sum of weight of examples, which are entailed by the preference graph represented by adjacency matrix A. Let S and S' be the subset of examples entailed by G and G', respectively. Because $E \subseteq E'$, the examples entailed by G can also be entailed by G', namely, $S \subseteq S'$. $g(A) \leq g(A')$. ■

Theorem 4. Let $B_v = (b_1, b_2, ...b_h, -1, ..., -1)$ be the vector associated with the node v at depth h in the search tree, the upper and lower bound at v are $g(\overline{A_v})$ and $g(\underline{A_v})$, respectively, where $\overline{A_v} = A(\overline{B_v}) = A(b_1, b_2, ..., b_h, \infty, ..., \infty)$ and $\underline{A_v} = A(\underline{B_v}) = A(b_1, b_2, ..., b_h, -1, ..., -1)$.

Proof: Let v' be a node in the subtree rooted at v with the associated vector B'. Let $A' = A(B')$. Let $\underline{E_v}$, E' and $\overline{E_v}$ be the sets of edges in preference graphs represented by vectors $\underline{B_v}$, B' and $\overline{B_v}$, then we have $\underline{E_v} \subseteq E' \subseteq \overline{E_v}$. By theorem 3, $g(\underline{A_v}) \leq g(A') \leq g(\overline{A_v})$. ■

If the upper bound of function $g(A)$ in formula (2) at node v is less than the lower bound at other nodes, the subtree rooted at v is pruned. By theorem 2, we can also prune the subtree rooted at v, if the corresponding preference graph of node v is inconsistent. If the lower bound is equal to the upper bound at node v, the values of function $g(A)$ at all nodes in the subtree rooted at v are equal, so we need not search the subtree rooted at v. We can implement the search as breadth-first search. Since there is only finite depth, the search is complete.

C. Transform Preference Graphs into CP-nets

The next step is to transform the obtained preference graph into a CP-net equivalently. The main task is to find the preferential dependency between variables and then generate the conditional preference tables for all variables according to the preferential dependency. Preferential dependency can be derived from the edges in preference graph. Given two values x_i^1 and x_i^2 of variable X_i, if difference assignments of the other variable change the preference relation between x_i^1 and x_i^2, this variable is one of the parents of X_i. Formally, given two values $x_i^1, x_i^2 \in D(X_i)$, we construct two sets of edges:

$$E_{x_i^1 \succ x_i^2} = \{(v_j, v_k) | (v_j, v_k) \in E, v_j[X_i] = x_i^2, v_k[X_i] = x_i^1\} \tag{11}$$

$$E_{x_i^2 \succ x_i^1} = \{(v_j, v_k) | (v_j, v_k) \in E, v_j[X_i] = x_i^1, v_k[X_i] = x_i^2\} \tag{12}$$

For a pair of edges $(v_j, v_k) \in E_{x_i^1 \succ x_i^2}$, $(v'_j, v'_k) \in E_{x_i^2 \succ x_i^1}$, if there exists a variable$X_p \neq X_i$ satisfying:

$$v_j[X_p] = v_k[X_p] \neq v'_j[X_p] = v'_k[X_p] \tag{13}$$

$$\begin{aligned} & v_j[V/(X_i \cup X_p)] = v_k[V/(X_i \cup X_p)] \\ = \quad & v'_j[V/(X_i \cup X_p)] = v'_k[V/(X_i \cup X_p)] \end{aligned} \tag{14}$$

X_p is one of the parent variables of X_i, $X_p \subseteq Pa(X_i)$.

In order to derive a CP-net from a preference graph, we first find the parents of every variable by checking every pair of edges, and then generate the conditional preference tables according to the preferential dependency. The procedure is summarized in Algorithm 1.

Algorithm 1. Transform Preference Graph into CP-net

Input: Preference graph G;
set $V = \{X_1, X_2, ..., X_n\}$ of variables;
$D(X_i), i = 1...n$;
Output: CP-net N;
for every pair of edges e', e'' **do**
  if $e' \in E_{x_i^k \succ x_i^l}$ and $e'' \in E_{x_i^l \succ x_i^k}$, $x_i^k, x_i^l \in D(X_i)$ **then**
    if exist X_j satisfying formula (13) and (14) **then**
      $X_j \subseteq Pa(X_i)$;
    end if
  end if
end for
for every edge (o', o), $o[X_i] \neq o'[X_i], o[V/X_i] = o'[V/X_i]$ **do**
  Add CPR "$o[Pa(X_i)] : o[X_i] \succ o'[X_i]$" to $CPT(X_i)$;
end for

D. Properties of the Learning Method

Now we prove the optimality of the proposed method. We first prove the optimality of the obtained preference graphs. Then we prove the preference graph and the CP-net transformed from it are equivalent.

Theorem 5. The preference graph obtained by the branch-and-bound search entails maximal satisfiable subset of examples.

Proof: Since there is only finite depth, the branch-and-bound search is complete, and the maximum of formula (2) must be obtained. According to the proof of Theorem 1, $th(e^A - I)$ is the preference relation entailed by preference graph, so $g(A)$ in formula (2) is the sum of weight of the examples entailed by the preference graph, namely, $g(A) = \sum_{(o_i \succ o'_i) \in P_G} w_i$. Thus the branch-and-bound search can get the preference graph entailing the maximal satisfiable subset of examples. ■

Definition 5. For CP-net N and preference graph G over a same set of variables, if for any two outcomes $x, y \in \Omega$, $x \succ_N y \Leftrightarrow x \succ_G y$, we say CP-net N and preference graph G are **equivalent**.

Theorem 6. Let N be the CP-net obtained from the preference graph G using Algorithm 1, for any two outcomes $o, o' \in \Omega$, if $o \succ_G o'$, $o \succ_N o'$.

Proof: If $o \succ_G o'$, there exists a path from o' to o in G. Let the nodes on the path be $o_1, o_2, ..., o_m$ sequentially, where $o_1 = o', o_m = o$. According to the definition of preference graph, o_i, o_{i+1} differ on only one variable. So we only need to prove for any two outcomes $o_j, o_k \in \Omega$ differing on only one variable, if $o_j \succ_G o_k$, $o_j \succ_N o_k$. Let $o_j[X_i] \neq o_k[X_i]$ and $o_j[V/X_i] = o_k[V/X_i]$. The conditional preference rule "$o_j[Pa(X_i)] : o_j[X_i] \succ o_k[X_i]$" will be generated from the edge (o_k, o_j) by Algorithm 1. This rule entails $o_j \succ_N o_k$. ■

Theorem 7. If a preference graph G is complete, the obtained CP-net N from G using Algorithm 1 is complete and equivalent to G.

Complete preference graph and complete CP-net are defined in Definition 2 and Definition 1. Now we prove complete preference graph can be translated into complete CP-net equivalently by Algorithm 1.

Proof: Firstly, we prove the completeness of N. If G is complete, then for any two values of a variable $x, x' \in D(X_i)$, the set of relevant edges $E_{x,x'} = E_{x \succ x'} \cup E_{x' \succ x}$ satisfies $\{o[V/X_i] | (o, o') \in E_{x,x'}\} = D(V/X_i) = \prod_{X_j \in V, X_j \neq X_i} D(X_j)$. Because $Pa(X_i) \subseteq V/X_i$, $\{o[Pa(X_i)] | (o, o') \in E_{x,x'}\} = D(Pa(X_i)) = \prod_{X_j \in Pa(X)} D(X_j)$. This means $CPT(X_i)$ is complete. Since CPTs on all variables are complete, the CP-net N is complete.

Secondly, we prove the equivalence. By theorem 6, for any $o, o' \in \Omega$, if $o \succ_G o'$, then $o \succ_N o'$. Now we prove if $o \succ_N o'$, then $o \succ_G o'$. If $o \succ_N o'$, there exists a improving flipping sequence: $o_1 \succ_N o_2 \succ_N ... \succ_N o_m$, where $o_1 = o, o_m = o'$, o_i, o_{i+1} differ on only one variable. We only need prove for any two outcomes $o_j, o_k \in \Omega$ differing on only one variable, if $o_j \succ_N o_k$, $o_j \succ_G o_k$. Let $o_j[X_i] \neq o_k[X_i]$ and $o_j[V/X_i] = o_k[V/X_i]$. Since $o_j \succ_N o_k$, there exists a conditional preference rule "$o_j[Pa(X_i)] : o_j[X_i] \succ o_k[X_i]$ " in N. Because the preference graph is complete, this rule is corresponding to an edge (o_k, o_j) in G, namely, $o_j \succ_G o_k$. ■

By Theorem 5 and Theorem 6 we obtain:

Corollary 2. The obtained CP-net entails the maximal satisfiable subset of examples.

E. Computational Complexity Analysis

In term of computational complexity, the branch-and-bound searches 2^k times in the worst case, where k, computed by formula (10), is the number of edges in preference graph.

Because of subtree pruning, the average search times are less than 2^k. In a preference graph with k edges, the number of pairs of edges is $k(k-1)/2$, so Algorithm 1 runs in time $o(k^2)$ in the worst case.

The computational complexity analysis indicates that our method can be applied to the datasets with small number of variables. Large real-life datasets with small number of variables, such as sushi dataset [19], are also suitable. In fact, the number of conditional preference rules in a CP-net is exponential respect to the number of variables. The problem of learning CP-nets from inconsistent examples is NP-hard. So we plan to find the approximation algorithm for this problem in further research. Although computational complexity of our method is relatively high, our method can obtain an optimal solution, which can be used to compare with the results obtained by other approximate methods.

F. Examples

The following examples illustrate our method.

Example 1. Let us consider the evening dress scenario in [16]. The set $V = \{J, S, P\}$ of variables stands for jacket, pants and shirt, respectively. The domains $D(J) = \{J_b, J_w\}$ stands for black jacket and white jacket, respectively; $D(P) = \{P_b, P_w\}$ stands for black pant and white pant, respectively; and $D(S) = \{S_r, S_w\}$ stands for red shirt and white shirt, respectively. The examples are shown in Table I.

TABLE I
EXAMPLES

(1)	$J_bP_bS_r \succ J_bP_bS_w$	(2)	$J_bP_bS_r \succ J_bP_wS_r$
(3)	$J_wP_bS_w \succ J_wP_wS_w$	(4)	$J_bP_bS_r \succ J_bP_wS_w$
(5)	$J_wP_bS_w \succ J_wP_bS_r$	(6)	$J_wP_wS_r \succ J_wP_wS_w$
(7)	$J_bP_bS_r \succ J_wP_bS_w$	(8)	$J_wP_bS_w \succ J_wP_wS_r$
(9)	$J_wP_bS_r \succ J_bP_bS_r$		

Observing the examples in Table I, we can find the example (5), (7) and (9) imply $J_bP_bS_r \succ J_bP_bS_r$ by transferring. So the examples are inconsistent. The indices of outcomes are shown in Table II.

TABLE II
INDICES OF OUTCOMES

Index	1	2	3	4
Outcome	$J_bP_bS_r$	$J_bP_bS_w$	$J_bP_wS_r$	$J_bP_wS_w$
Index	5	6	7	8
Outcome	$J_wP_bS_r$	$J_wP_bS_w$	$J_wP_wS_r$	$J_wP_wS_w$

Let the weight of the every example be 1, the training matrix A_P is shown in Fig. 2 a). Using branch-and-bound search, we obtain a preference graph shown in Fig. 2 c) with the adjacency matrix shown in Fig. 2 b).

Next, we translate the obtained preference graph into a CP-net using Algorithm 1. For variable J, since $E_{J_w \succ J_b} = \emptyset$, $Pa(J) = \emptyset$. Similarly, since $E_{P_w \succ P_b} = \emptyset$, $Pa(P) = \emptyset$. For the variable S, we have $E_{S_r \succ S_w} = \{(J_bP_bS_w, J_bP_bS_r), (J_wP_wS_w, J_wP_wS_r)\}$ and $E_{S_w \succ S_r} = \{(J_wP_bS_r, J_wP_bS_w), (J_bP_wS_r, J_bP_wS_w)\}$. For the two pairs of edges $((J_bP_bS_w, J_bP_bS_r), (J_wP_bS_r, J_wP_bS_w))$ and $((J_wP_wS_w, J_wP_wS_r), (J_bP_wS_r, J_bP_wS_w))$, variable J satisfies formula (13) and (14), so $J \subseteq Pa(S)$. Similarly, For the two pairs of edges $((J_wP_wS_w, J_wP_wS_r), (J_wP_bS_r, J_wP_bS_w))$ and $((J_bP_bS_w, J_bP_bS_r), (J_bP_wS_r, J_bP_wS_w))$, variable P satisfies formula (13) and (14), so $P \subseteq Pa(S)$. $Pa(S) = \{J, P\}$.

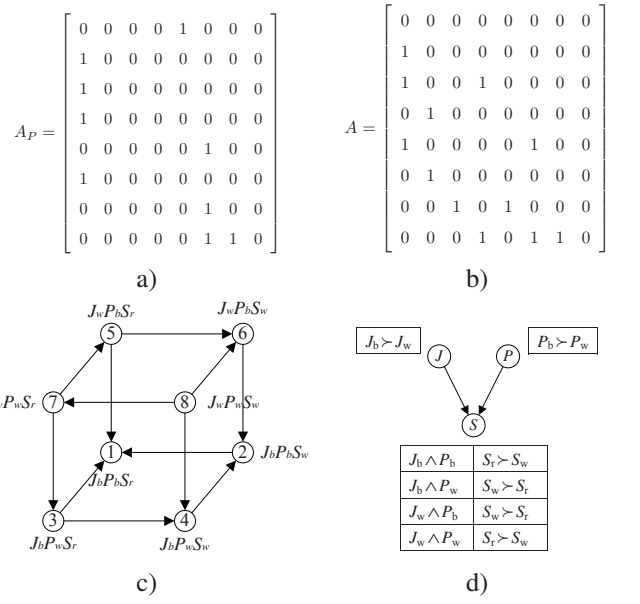

Fig. 2. Training matrix a), adjacency matrix b) of obtained preference graph c) and the final CP-net d).

By the preferential dependency, we can generate conditional preference tables for all variables. The final CP-net is shown in Fig. 2 d). It is easy to validate the preference graph and the CP-net in Fig. 2 are equivalent. Furthermore, this CP-net entails all of the training examples in Table I except example (9), which can be regarded as noise in examples. Thus the CP-net in Fig. 2 d) entails maximal satisfiable subset of examples.

Here, we give another example to illustrate our method can also handle multivalued variables.

Example 2. Let's consider the dinner scenario. There are three variables M, S and W standing for main course, soup and wine. One day, there were fish course (M_f), chicken course (M_c), fish soup (S_f), vegetable soup (S_v), white wine (W_w) and red wine (W_r) available. Bob chose fish course (M_f), vegetable soup (S_v) and red wine (W_r). The observed examples were $M_fS_vW_r \succ \{M_fS_vW_w, M_fS_fW_r, M_fS_fW_w, M_cS_vW_r, M_cS_vW_w, M_cS_fW_r, M_cS_fW_w\}$. Because Bob satisfied this dinner, the weights of examples were set to 3. The other day, there were pork course (M_p), chicken course (M_c), fish soup (S_f), vegetable soup (S_v), white wine (W_w) and red wine (W_r) available. Bob chose chicken course (M_c), vegetable soup (S_v) and red wine (W_r). The observed examples were $M_cS_cW_r \succ \{M_cS_fW_r, M_cS_vW_w, M_cS_fW_w, M_pS_fW_w, M_pS_fW_r, M_pS_vW_w, M_pS_vW_r\}$. Bob satisfied this dinner very much, the weights of the examples were set to 5. The

third day, there were fish course (M_f), chicken course (M_c), pork course (M_p), fish soup (S_f), vegetable soup (S_v) and only red wine (W_r) available. Bob chose pork course (M_p), vegetable soup (S_v) and red wine (W_r). The observed examples were $M_pS_vWr \succ \{M_fS_fW_r, M_fS_vW_r, M_cS_fW_r, M_cS_vW_r, M_pS_fW_r\}$. But Bob did not like this dinner, the weights of the examples are set to 1. The domains of variables are $D(M) = \{M_c, M_p, M_f\}$, $D(S) = \{S_v, S_f\}$ and $D(W) = \{W_r, W_w\}$. The indices of outcomes are shown in Table III.

TABLE III
INDICES OF OUTCOMES

Index	1	2	3	4
Outcome	$M_fS_fW_r$	$M_fS_fW_w$	$M_fS_vW_r$	$M_fS_vW_w$
Index	5	6	7	8
Outcome	$M_cS_vW_r$	$M_cS_fW_w$	$M_cS_fW_r$	$c_fS_fW_w$
Index	9	10	11	12
Outcome	$M_pS_fW_r$	$M_pS_fW_w$	$M_pS_vW_r$	$M_pS_vW_w$

The training matrix A_p is shown in Fig. 3 a). Using branch-and-bound search, we obtain a preference graph shown in Fig. 3 c) with the adjacency matrix shown in Fig. 3 b). Note this preference graph entails all the training examples except $M_pS_vW_r \succ M_fS_vW_r$ and $M_pS_vW_r \succ M_cS_vW_r$, which can be regarded as noise.

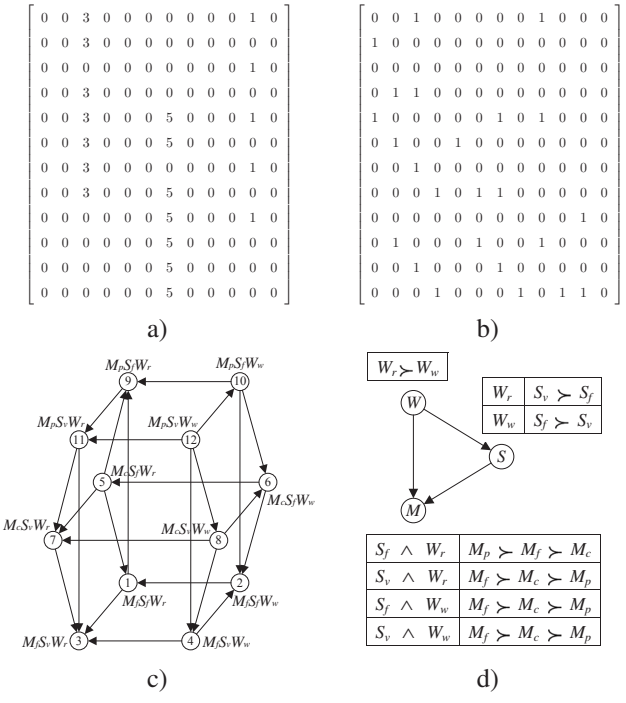

Fig. 3. Training matrix a), adjacency matrix b) of obtained preference graph c) and the final CP-net d).

Next, we translate this preference graph into CP-net. For variable W, $E_{W_w \succ W_r} = \emptyset$, so $Pa(W) = \emptyset$. For variable S, $E_{S_f \succ S_v}$=$\{(M_fS_vW_r, M_fS_fW_r), (M_cS_vW_r, M_pS_fW_r), (M_cS_vW_r, M_pS_fW_r)\}$, and $E_{S_v \succ S_f}$=$\{(M_fS_fW_w, M_fS_vW_w), (M_cS_fW_w, M_cS_vW_w), (M_pS_fW_w, M_pS_vW_w)\}$. For the pair of edges $(M_fS_vW_r, M_fS_fW_r)$ and $(M_fS_fW_w, M_fS_vW_w)$, variable W satisfies formula (13) and (14). No other variables satisfy these two formulas, so $Pa(S) = \{W\}$. By the same way, we can get $Pa(M) = \{S, W\}$. Conditional preference tables can be obtained according to the preferential dependency. The entire CP-net is shown in Fig. 3 d).

V. EXPERIMENTAL RESULTS AND DISCUSSION

In order to verify our model and algorithm's correctness and accuracy, we set out to put the algorithm into practice on simulated data and real data. In first experiment, we generate pairwise comparisons with noise as learning examples from a CP-net randomly, and then learn CP-nets from these examples. In second experiment, real users' preferences collected by Kamishima [19] are used to generate the learning samples. The dataset contains 100 kinds of sushi and 5000 users' preferences orders on these sushi. In third experiment, we verify our method on 4 regression datasets (computer hardware, red wine quality, white wine quality and concrete compressive strength) from UCI repository [40] . These datasets contain outcome (item) features and the corresponding outputs (e.g. quality in wine quality dataset, performance in computer hardware dataset and strength in concrete compressive strength dataset.). The outputs are treated as preference orders. In the experiments, we compare the method in [11], RankNet [28], AdaRank [38], RankBoost [22] and Coordinate Ascent [39] with ours. The details are explained as follows.

A. Experimental Results on Simulated Data

First, we randomly generate preference graph of an acyclic CP-net with three binary attributes to represent a user's preference. Second, pairwise comparisons of outcomes entailed by the preference graph are generated randomly. Considering that the user may not make every decision exactly agree with her/his preference, we add some noise to the learning examples. Namely, we change the preference relation between two outcomes at a probability of p, which is called noise level. Finally, a preference graph and CP-net are learned using our method.

In order to measure the correctness of the rebuilt CP-net, we compute the similarity between preference graphs as the similarity between CP-nets, for it's not easy to compare two CP-net directly. The similarity between two preference graphs is defined as:

$$similarity = \frac{num(agree\_edge)}{num(total\_edge)} \tag{15}$$

The numerator is the number of edges which have same start node and end node and same direction in those two preference graphs. The denominator is the total number of edges in preference graph. Because the two preference graphs are over the same outcome space, the numbers of edges in them are same. We also define the agreement between rebuilt CP-net and learning examples:

$$agreement = \frac{num(agree\_example)}{num(total\_example)} \tag{16}$$

The numerator is the number of examples entailed by the rebuilt CP-net. The denominator is the total number of examples.

We generate 20, 50, 100, 200, 500 and 1000 pairwise comparisons at noise level 0, 0.01, 0.05, 0.1, 0.2 and 0.4 respectively. For each configuration, we repeat 1000 times, and calculate the average similarity between initial CP-net and rebuilt CP-net and the average agreement between rebuilt CP-net and learning examples. The results are shown in Fig. 4. Fig. 4 a) represents the average similarity between initial CP-net and rebuilt CP-net at different noise level and learning examples size. Fig. 4 b) represents the average agreement between the rebuilt CP-net and the learning examples at different noise level and learning examples size.

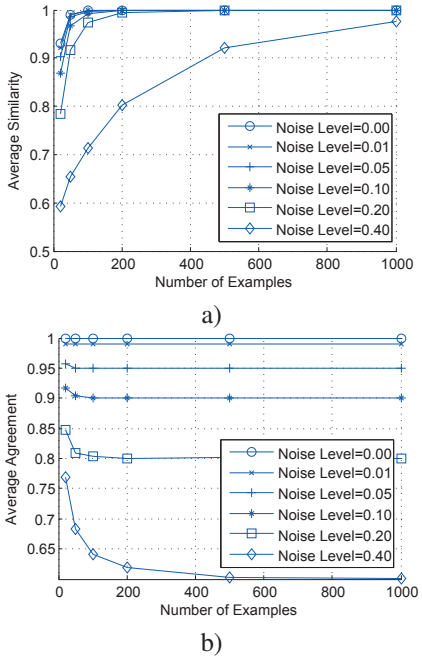

Fig. 4. The average similarity a) and the average agreement b) at different noise level and learning examples size on simulated data.

In Fig. 4 a) we can see at the same noise level, as the number of examples increases, the average similarity between the rebuilt CP-net and the initial CP-net increases and converges at 1. That means the rebuilt CP-net will approach the initial CP-net as the number of examples increases, even at high noise level. The converging speed is low at high noise level, at this time, more examples should be used to get more accurate result. While in Fig. 4 b) we can see at noise level p the average sample agreement decreases and converges at $1-p$, as the number of examples increases. The reason is when the number of examples is small, there are less conflicts in the learning examples, and easy to be satisfied. The convergency of average agreement means the noisy examples are removed by our method, and the rest of the examples are entailed by the rebuilt CP-net.

To the best of our knowledge, there does not exist study in learning CP-nets from inconsistent examples, we compare our method with the method proposed in [11]. Similar to our method, method in [11] learns CP-nets passively, but it cannot handle noisy examples. In the experiments, we fix noise level at 0. [11]'s method also requires that the training examples are transparent entailed by CP-nets(see Definition 5 in [11]), otherwise, [11]'s method fails to return a CP-net. We generate 20, 50, 100, 200, 500 and 1000 noiseless pairwise comparisons transparent entailed by CP-nets. CP-nets are rebuilt using [11]'s method and ours. For each size of examples, the experiments are repeated 1000 times. The average similarity between rebuilt CP-net and initial CP-net and the average sample agreement are compared separately. The results are shown in Fig. 5. Our method and [11]'s method get similar results at noise level 0. the CP-nets rebuilt by both of methods approach the initial CP-net perfectly, and both of them satisfy all the learning examples correctly. Note the results in Fig.5 are obtained on the training samples transparent entailed by CP-nets. If this requirement is not met, [11]'s method can not return a CP-net. While, our method can always obtain a CP-net. Table IV shows the success rate of [11]'s method and our method on noiseless training samples which is not satisfied transparent entailment. We can observe that [11]'s method is more likely to fail by given more training samples. Compared with [11]'s method, there are less constraints on training samples in our method. The other advantage of our method is that our method can handle noisy examples, while [11]'s method can not. Our method can be applied more widely than the method proposed in [11].

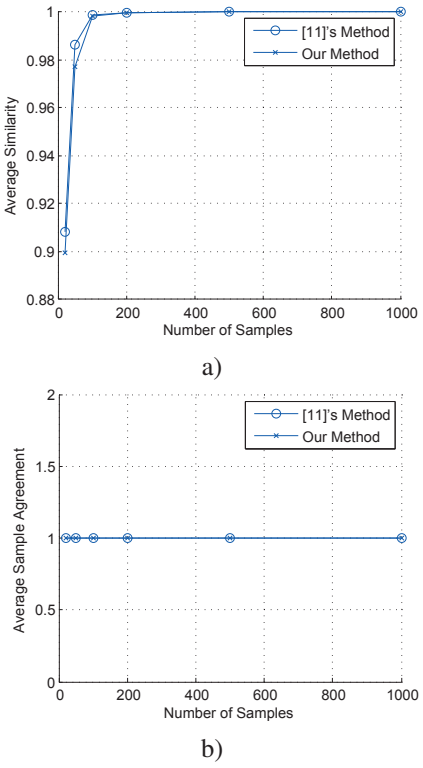

Fig. 5. Comparison our method with [11]'s on simulated data.

We compare our method with RankNet [28], AdaRank [38], RankBoost [22] and Coordinate Ascent [39] on simulated data. These method are the most popular methods for learning to rank in recent years, and can handle noise training samples.

TABLE IV
COMPARISON SUCCESS RATE OF OUR METHOD WITH THAT OF [11]'S METHOD ON SIMULATED DATA

Sample size	20	50	100	200	500	1000
Success rate of [11]'s method	0.916	0.808	0.788	0.774	0.764	0.761
Success rate of our method	1.000	1.000	1.000	1.000	1.000	1.000

The results, shown in Table V and Table VI, indicate that our method outperforms these methods. The reason is that the methods for learning to rank, such as RankNet [28], AdaRank [38], RankBoost [22] and Coordinate Ascent [39], can't represent conditional preference perfectly.

We evaluate the complexity of branch-and-bound search. For different number of edges in preference graphs, we generate learning examples at noise level 0, 0.01, 0.05, 0.1, 0.2 and 0.4 respectively. We repeat the experiment 1000 times and calculate the average number of iterations of branch-and-bound search. The results are shown in Fig. 6, in which 'REF' denotes the number of nodes in search tree in Fig. 1. In Fig. 6, we can see that the branch-and-bound search reduces a great number of search.

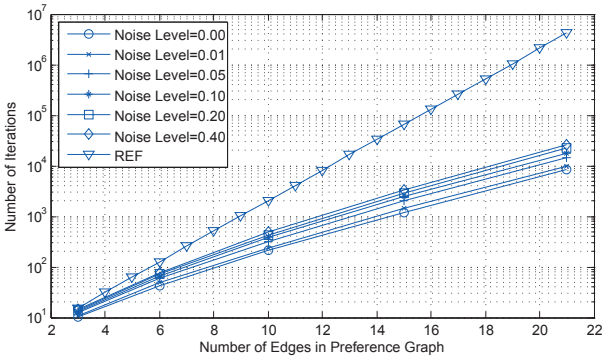

Fig. 6. Average Number of Iterations of Branch-and-Bound Search.

B. Experimental Results on Sushi Dataset

Sushi dataset [19] collected by Kamishima et al. contains 100 kinds of sushi and 5000 users' preferences on these sushi. Each kind of sushi is described by 7 features. Each people was required to order 10 favorite sushi. More details can be found in [19]. To simplify the problem and reduce the time consuming, we only focus on three features: the style, the heaviness and normalized price. These features are considered to be the most important features which affect user's preference based on common sense. "The style" feature is binary. 0 denotes "maki" style, and 1 denotes other style. While the other two features are continuous. Because CP-net cannot process continuous value, we choose to use the average value of the features to discretize them. All the data lower than the average value is set to 0, all the data higher than average value is set to 1.

After preprocessing, we classify all the sushi into $2^3 = 8$ types, and turn the user's preference order on different sushi into preference on the 8 different types. Knowing the preference order on those types, we generate pairwise comparisons of sushi with noise as learning examples. For each user's preference order, we generate 20, 50, 100, 200, 500 and 1000 pairwise comparisons of sushi at noise level 0, 0.01, 0.05, 0.1, 0.2 and 0.4 respectively. For each configuration, we choose 1000 users randomly to repeat the experiment. The similarity between the user's preference order and the rebuilt CP-net is defined as:

$$similarity = \frac{|\{o, o' | o \succ_N o', o \succ_{PO} o'\}|}{|\{o, o' | o \succ_{PO} o'\}|} \tag{17}$$

where, $o, o' \in \Omega$, N denotes the rebuilt CP-net, PO denotes the preference order. The numerator is the number of pairwise comparisons entailed by both CP-net N and preference order PO. The denominator is the number of pairwise comparisons entailed only by preference order PO.

We compute the average sample agreement defined by formula (16). The results are shown in Fig. 7 . Comparing with the first experiment, we can find out that the trend of the curves is similar. So all the conclusions we get from the first experiment are still make sense. Besides the similarity, there are also some differences between the two experiments. CP-net can only represents partial order, while the users' preferences in sushi dataset are total orders, so there are some preferences can not be satisfied by CP-nets. For this reason, although we set noise to 0 and use 1000 examples, the agreement and similarity cannot approach 1. And the curves are lower than that of the first experiment.

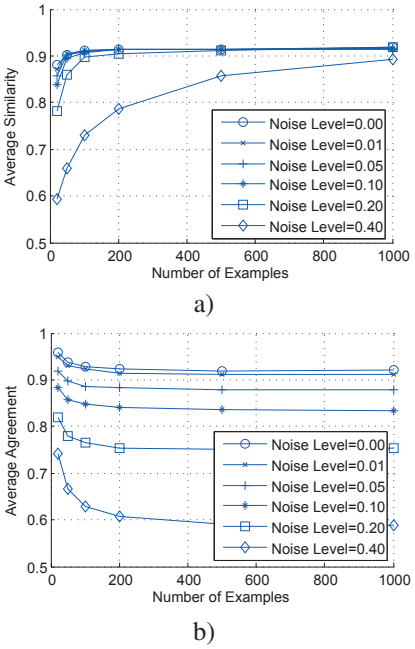

Fig. 7. The average similarity a) and the average agreement b) at different noise level and sample size on sushi dataset.

We compare [11]'s method with ours on sushi dataset. Because the method in [11]assumes the training samples are noiseless and transparent entailed by CP-nets, this method can only handle some of the data at noise level $p = 0$. So we pick up the users' preference orders that satisfy the

TABLE V

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON SIMULATED DATA(SAMPLE SIZE=50)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.5318	0.5431	0.5285	0.5451	0.5441	0.5548	0.5315	0.5332	0.5248	0.5175
RankBoost	0.8592	0.8649	0.8610	0.8637	0.8593	0.8487	0.8618	0.8356	0.8498	0.7996
AdaRank	0.6975	0.6501	0.6932	0.6486	0.6998	0.6450	0.7128	0.6444	0.7246	0.6501
Coordinate Ascent	0.7115	0.8434	0.7088	0.8375	0.7075	0.8160	0.7063	0.7871	0.6804	0.7312
Our Method	0.9895	1.0000	0.9892	0.9905	0.9842	0.9506	0.9669	0.9038	0.9170	0.8092

TABLE VI

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON SIMULATED DATA(SAMPLE SIZE=100)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.5693	0.5933	0.5773	0.6024	0.5670	0.5760	0.5580	0.5686	0.5448	0.5381
RankBoost	0.8583	0.8561	0.8596	0.8570	0.8597	0.8395	0.8619	0.8302	0.8555	0.7928
AdaRank	0.7054	0.6504	0.7146	0.6551	0.7172	0.6426	0.7238	0.6514	0.7407	0.6513
Coordinate Ascent	0.7131	0.8302	0.7172	0.8276	0.7207	0.8066	0.7193	0.7805	0.6973	0.7201
Our Method	0.9994	1.0000	0.9988	0.9901	0.9973	0.9496	0.9919	0.9000	0.9747	0.8028

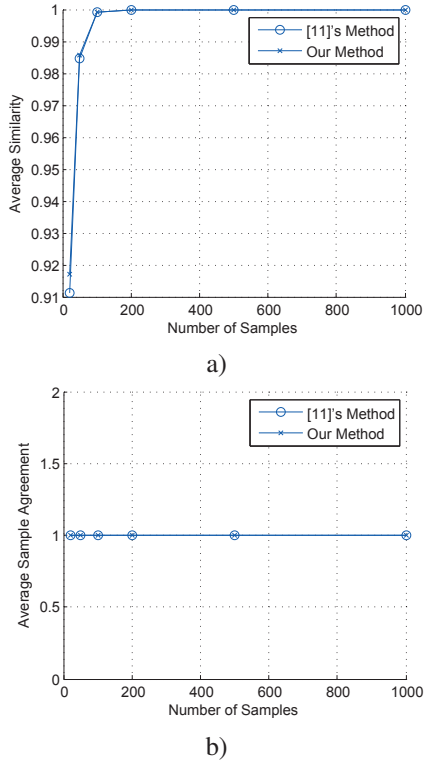

Fig. 8. Comparison our method with [11]'s on Sushi dataset.

assumption, and generate training data using these preference orders. The results are shown in Fig. 8. On the selected data of sushi dataset, the results of our method and [11]'s method are very similar as that in the first experiment. If we select user preference orders randomly, [11]'s method will fail in most of the cases. Table VII shows success rate of [11]'s method and that of our method on randomly selected user preference orders in sushi dataset. Similar as on simulated data, success rate of [11]'s method is lower on larger training samples. Compared with [11]'s method, the advantage of our method is that there are less constraints on training data, and our method can be applied widely.

We compare our method with RankNet [28], AdaRank [38],RankBoost [22] and Coordinate Ascent [39] on sushi dataset. The results are shown in Table VIII and Table IX.

TABLE VII

COMPARISON SUCCESS RATE OF OUR METHOD WITH THAT OF [11]'S METHOD ON SUSHI DATASET

Sample size	20	50	100	200	500	1000
Success rate of [11]'s method	0.229	0.064	0.058	0.053	0.048	0.044
Success rate of our method	1.000	1.000	1.000	1.000	1.000	1.000

Because these methods can't represent conditional preference perfectly, similar to the results on simulated data, our method outperforms them.

C. Experimental Results on Regression Datasets

Because of lack of suitable benchmark datasets, we evaluate our approach on 4 benchmark datasets for regression (computer hardware, red wine quality, white wine quality and concrete compressive strength) from UCI repository [40]. These datasets contain several features of outcome (item). We select first 3 principal components as the new features after applying PCA to the original features in these datasets. We choose to use the average values of the new features to discretize them. 16 outcomes(items) are randomly selected from each dataset, and total orders of these outcomes are built by sorting the output of the datasets (e.g. quality in wine quality dataset, performance in computer hardware dataset and strength in concrete compressive strength dataset). These total orders are treated as preference orders stated by some users. We construct 1000 orders for each dataset and generate 20, 50, 100, 200, 500 and 1000 training examples at noise level 0, 0.01, 0.05, 0.1, 0.2, 0.4 according to each order by the same way on sushi dataset. We compute the similarity between the orders and the rebuilt CP-nets using formula (17). The sample agreement are also computed. The results on these datasets are shown in Fig. 9-Fig.12. the curves are similar to that in Fig. 7. So we can draw the same conclusions on these datasets.

We compare our method with learning to rank methods on these dataset. The results are presented in Table X-Table XIII. Our method outperforms the learning to rank methods. We don't present the results of the method in [11], which fails in most of the cases on these datasets. The reason is that most the training examples generated on these datasets don't satisfied the constraint in [11].

TABLE VIII

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON SUSHI DATASET(SAMPLE SIZE=50)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.5488	0.5482	0.5528	0.5565	0.5431	0.5450	0.5418	0.5391	0.5322	0.5204
RankBoost	0.8594	0.8726	0.8630	0.8710	0.8582	0.8504	0.8526	0.8246	0.8553	0.7951
AdaRank	0.6788	0.6853	0.6753	0.6813	0.6851	0.6799	0.6857	0.6730	0.6854	0.6625
Coordiante Ascent	0.8330	0.8639	0.8304	0.8559	0.8318	0.8366	0.8242	0.8038	0.7961	0.7449
Our Method	0.9033	0.9368	0.9052	0.9307	0.9018	0.8966	0.8958	0.8569	0.8600	0.7791

TABLE IX

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON SUSHI DATASET(SAMPLE SIZE=100)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.5900	0.5926	0.5903	0.5908	0.5886	0.5829	0.5816	0.5684	0.5501	0.5340
RankBoost	0.8624	0.8680	0.8665	0.8679	0.8635	0.8487	0.8645	0.8276	0.8503	0.7789
AdaRank	0.6803	0.6852	0.6770	0.6777	0.6828	0.6771	0.6889	0.6735	0.6787	0.6623
Coordiante Ascent	0.8429	0.8570	0.8444	0.8524	0.8419	0.8288	0.8346	0.7953	0.8121	0.7288
Our Method	0.9106	0.9291	0.9114	0.9226	0.9083	0.8866	0.9074	0.8475	0.8971	0.7658

TABLE X

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON COMPUTER HARDWARE DATASET(SAMPLE SIZE=100)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.6063	0.6093	0.5945	0.5971	0.5817	0.5779	0.5833	0.5699	0.5515	0.5366
RankBoost	0.9041	0.9116	0.9001	0.9010	0.8984	0.8764	0.8946	0.8449	0.8824	0.7815
AdaRank	0.7657	0.7702	0.7669	0.7671	0.7683	0.7516	0.7655	0.7292	0.7239	0.6796
Coordiante Ascent	0.8505	0.8679	0.8512	0.8637	0.8448	0.8364	0.8337	0.8022	0.7983	0.7412
Our Method	0.9745	0.9479	0.9745	0.9412	0.9741	0.9073	0.9729	0.8631	0.9677	0.7752

TABLE XI

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON RED WINE QUALITY DATASET(SAMPLE SIZE=100)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.5837	0.5863	0.5711	0.5750	0.5694	0.5671	0.5617	0.5539	0.5432	0.5290
RankBoost	0.8089	0.8175	0.8077	0.8106	0.8052	0.7977	0.8076	0.7838	0.8064	0.7559
AdaRank	0.6235	0.6281	0.6147	0.6169	0.6185	0.6244	0.6185	0.6245	0.6020	0.6269
Coordiante Ascent	0.8149	0.8390	0.8157	0.8321	0.8148	0.8101	0.8107	0.7799	0.7907	0.7207
Our Method	0.9329	0.8952	0.9326	0.8896	0.9311	0.8583	0.9292	0.8228	0.9217	0.7483

VI. CONCLUSIONS AND FUTURE WORK

Within artificial intelligence, the problem of preference learning or preference elicitation is valuable in theory and application. In this paper, we propose the model of learning CP-nets from inconsistent examples and present a method to solve this model. We do not learn the CP-nets directly. Instead, we first learn the preference graph in which dominance testing and consistency testing are easy. And then we transform the obtained preference graph into CP-net equivalently. We prove the obtained CP-net entails the maximal satisfiable subset of examples.

Improving the performance of the learning method is the main further research. Since the problem of learning CP-nets from inconsistent examples is NP-hard, we need to find the approximation algorithm. In other further research direction, we plan to research the learning methods for other CP-nets classes, including TCP-nets [3]and Stronger Conditional Preference Statements [6], [7], which are more expressive than CP-nets. Finally, online learning is another research direction, where the learner receives the examples one by one, and each time the learner updates the user's preference.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (61173120, 60903096) and Natural Science Foundation of Hebei Province (F2009001435).

REFERENCES

[1] Boutilier, C., et al. Reasoning With Conditional Ceteris Paribus Preference Statements. in 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI '99). 1999.

[2] Boutilier, C., F. Bacchus, and R.I. Brafman. UCP-Networks: A Directed Graphical Representation of Conditional Utilities. in 17th Annual Conference on Uncertainty in Artificial Intelligence (UAI '01) 2001.

[3] Brafman, R.I., C. Domshlak, and S.E. Shimony, On Graphical Modeling of Preference and Importance. Journal of Artificial Intelligence Research, 2006. 25: p. 398-424.

[4] Chatel, P., I. Truck, and J. Malenfant, LCP-Nets: A Linguistic Approach for Non-functional Preferences in a Semantic SOA Environment. Journal of Universal Computer Science, 2010. 16(1): p. 198-217.

[5] Kaci, S. and H. Prade. Relaxing Ceteris Paribus Preferences with Partially Ordered Priorities. in ECSQARU 2007. 2007.

[6] Wilson, N. Extending CP-Nets with Stronger Conditional Preference Statements. in 19th National Conference on Artificial Intelligence (AAAI '04). 2004. San Jose, CL.

[7] Wilson, N., Computational techniques for a simple theory of conditional preferences. Artificial Intelligence, 2011. 175(7-8): p. 1053-1091.

[8] Yaman, F. and M. desJardins. More-or-Less CP-Networks. in 24th Annual Conference on Uncertainty in Artificial Intelligence (UAI '08). 2008.

[9] Koriche, F. and B. Zanuttini. Learning Conditional Preference Networks with Queries. in 21st International Joint Conference on Articial Intelligence(IJCAI '09). 2009.

[10] Koriche, F. and B. Zanuttini, Learning Conditional Preference Networks. Artificial Intelligence, 2010. 174: p. 685-703.

[11] Dimopoulos, Y., L. Michael, and F. Athienitou. Ceteris Paribus Preference Elicitation with Predictive Guarantees. in 21st International Joint Conference on Artificial Intelligence (IJCAI '09). 2009.

[12] Lang, J. and J. Mengin. Learning Preference Relations over Combinatorial Domains. in NMR'08. 2008.

[13] Lang, J. and J. Mengin. The complexity of learning separable ceteris paribus preferences. in 21st International Joint Conference on Artificial Intelligence (IJCAI '09). 2009.

TABLE XII

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON WHITE WINE QUALITY DATASET(SAMPLE SIZE=100)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.5584	0.5615	0.5635	0.5663	0.5521	0.5522	0.5532	0.5474	0.5378	0.5265
RankBoost	0.8241	0.8343	0.8266	0.8359	0.8173	0.8228	0.8253	0.8261	0.8093	0.8035
AdaRank	0.5587	0.5541	0.5617	0.5578	0.5546	0.5630	0.5550	0.5730	0.5569	0.5996
Coordiante Ascent	0.7787	0.8084	0.7824	0.8064	0.7810	0.7844	0.7677	0.7537	0.7591	0.7042
Our Method	0.9287	0.8882	0.9273	0.8805	0.9257	0.8495	0.9250	0.8154	0.9163	0.7447

TABLE XIII

COMPARISON THE PROPOSED METHOD WITH LEARNING TO RANK METHODS ON CONCRETE COMPRESSIVE STRENGTH DATASET(SAMPLE SIZE=100)

Noise Level(p)	0		0.01		0.05		0.10		0.20	
Method	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement	Similarity	Agreement
RankNet	0.5850	0.5867	0.5886	0.5908	0.5788	0.5747	0.5757	0.5652	0.5569	0.5382
RankBoost	0.8704	0.8768	0.8655	0.8670	0.8704	0.8497	0.8685	0.8225	0.8638	0.7667
AdaRank	0.7362	0.7430	0.7347	0.7392	0.7411	0.7284	0.7375	0.7120	0.7190	0.6830
Coordiante Ascent	0.8349	0.8549	0.8373	0.8505	0.8303	0.8219	0.8204	0.7904	0.8018	0.7346
Our Method	0.9423	0.9096	0.9420	0.9026	0.9394	0.8694	0.9380	8298	0.9275	0.7560

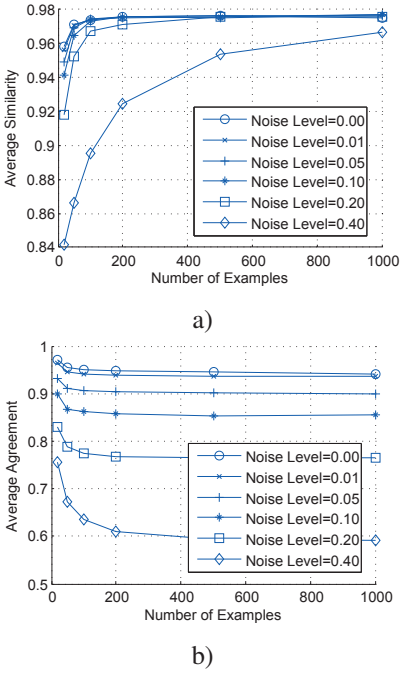

Fig. 9. The average similarity a) and the average agreement b) at different noise level and sample size on computer hardaware set.

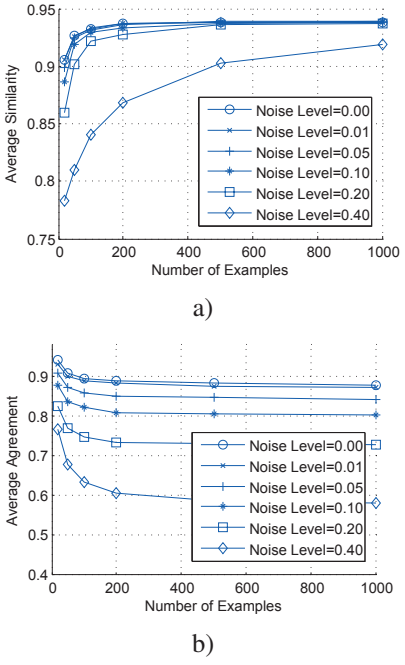

Fig. 10. The average similarity a) and the average agreement b) at different noise level and sample size on red wine quality dataset.

[14] Goldsmith, J., et al., The Computational Complexity of Dominance and Consistency in CP-Nets. Journal of Artificial Intelligence Research, 2008. 33: p. 403-432.

[15] Goldsmith, J., et al. The Computational Complexity of Dominance and Consistency in CP-nets. in 19th International Joint Conference on Artificial Intelligence (IJCAI '05). 2005.

[16] Boutilier, C., et al., CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements. Journal of Articial Intelligence Research, 2004. 21: p. 135-191.

[17] Chevaleyre, Y., et al., Learning Ordinal Preferences on Multiattribute Domains: the Case of CP-nets. 2011, Springer-Verlag: Berlin. p. 273-296.

[18] Moler, C. and C.V. Loan, Nineteen Dubious Ways to Compute the Exponential of a Matrix. SIAM Review, 1978. 20(4): p. 801-836.

[19] T. Kamishima, "Nantonac Collaborative Filtering: Recommendation Based on Order Responses", KDD2003, pp.583-588 (2003)

[20] W.W. Cohen, R.E. Schapire and Y. Singer. Learning to order things. Journal of Artificial Intelligence Research, 10:243-270, 1999.

[21] T.Kamishima et. al. "Supervised Ordering - An Empirical Survey", ICDM2005

[22] Y.Freund et. al. "An Efficient Boosting Algorithm for Combining Preferences", JMLR, vol.4 (2003)

[23] H.Kazawa et. al. "Order SVM: a kernel method for order learning based on generalized order statistics", Systems and Computers in Japan, vol.36 (2005)

[24] R.Herbrich et. al. "Learning Preference Relations for Information Retrieval", ICML1998 Workshop: Text Categorization and Machine Learning

[25] T.Joachims, "Optimizing Search Engines Using Click through Data", KDD2002

[26] Schmitt, M., Martignon, L.,. On the complexity of learning lexicographic strategies. Journal of Machine Learning Research vol. 7, pp. 55-83,2006

[27] F. Yaman et al., Democratic approximation of lexicographic preference models, Artificial Intelligence (2010), doi:10.1016/j.artint.2010.11.012

[28] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, G. Hullender, "Learning to Rank using Gradient Descent", 22nd International Conference on Machine Learning, Bonn, 2005.

[29] Mingfeng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. FRank: A Ranking Method with Fidelity Loss, SIGIR 2007.

[30] Tao Qin, Tie-Yan Liu, Wei Lai, Xu-Dong Zhang, De-Sheng Wang, and Hang Li.Ranking with Multiple Hyperplanes, SIGIR 2007.

[31] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In ICML, 2007.

[32] Carvalho, V., Elsas, J., Cohen, W., Carbonell, J. A meta-learning

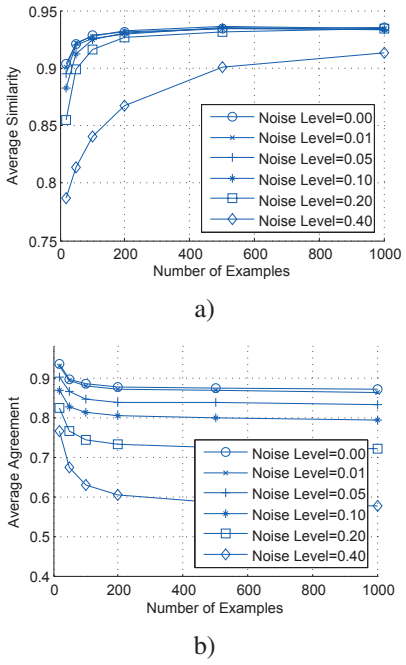

Fig. 11. The average similarity a) and the average agreement b) at different noise level and sample size on white wine quality dataset.

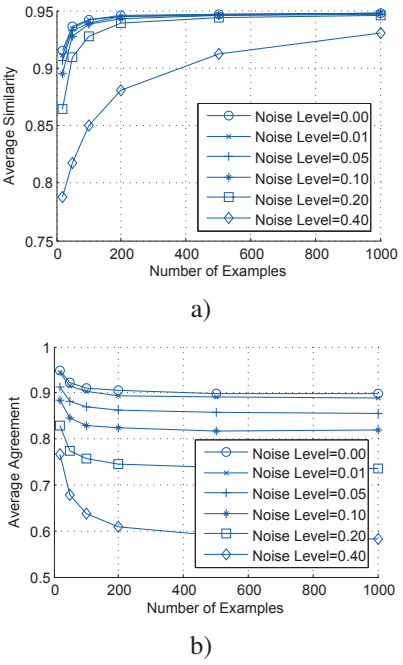

Fig. 12. The average similarity a) and the average agreement b) at different noise level and sample size on concrete compressive strength dataset.

approach for robust rank learning. In SIGIR 2008 workshop on learning to rank for information retrieval, 2008.
[33] Fen Xia, Tie-Yan Liu, Hang Li, Statistical Consistency of Top-k Ranking, NIPS 2009.
[34] C.J.C. Burges, K.M. Svore, P.N. Bennett, A. Pastusiak and Q. Wu, "Learning to Rank Using an Ensemble of Lambda-Gradient Models", Journal of Machine Learning Research: Workshop and Conference Proceedings, vol. 14, pp. 25-35, 2011.
[35] C.J.C. Burges, R. Ragno and Q.V. Le. Learning to Rank with Non-Smooth Cost Functions.Advances in Neural Information Processing Systems, 2006.
[36] Q. Wu, C. Burges, K. M. Svore, and J. Gao. Adapting Boosting for Information Retrieval Measures. Information Retrieval, 2009.
[37] J.H. Friedman. Greedy function approximation: A gradient boosting machine. Annals of Statistics, 2001.
[38] J. Xu and H. Li. AdaRank: a boosting algorithm for information retrieval. In Proc. of SIGIR, pages 391-398, 2007.
[39] D. Metzler and W.B. Croft. Linear feature-based models for information retrieval. Information Retrieval, 10(3): 257-274, 2000.
[40] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

Juntao Liu received the BS and MS degrees in Computer Science from Ordnance Engineering College, Shijiazhuang, China, in 2002 and 2005, respectively. He is a lecturer in Department of Computer Engineering, Ordnance Engineering College. He is currently pursuing the Ph.D. degree in Department of Electronics and Information Engineering, Huazhong University of Science & Technology. His research interests include data mining, machine learning and computer vision.

Yi Xiong received the BS degree in Communication and Information System from Huazhong University of Science & Technology (HUST), Wuhan, China, in 2011. Now she is graduate students in Department of Electronics and Information Engineering, Huazhong University of Science & Technology. Her research interests include data mining and machine learning.

Caihua Wu received the BS, MS and PhD degrees in Computer Science from Ordnance Engineering College, Shijiazhuang, China, in 2003, 2006 and 2009 respectively. Now she is a lecturer in Department of Information Counterwork, Air Force Radar Academy. Her research interests include data mining, information counterwork and software engineering.

Zhijun Yao received the BS and MS degrees in Communication and Information System from Huazhong University of Science & Technology (HUST), Wuhan, China, in 2000 and 2007, respectively. He is currently pursuing the PhD degree in Department of Electronics and Information Engineering, Huazhong University of Science & Technology. His research interests include: image processing, pattern recognition and computer vision.

Wenyu Liu received the BS degree in Computer Science from Tsinghua University, Beijing, China, in 1986, and the MS and PhD degrees, both in Electronics and Information Engineering, from Huazhong University of Science & Technology (HUST), Wuhan, China, in 1991 and 2001, respectively. He is now a professor and associate dean of the Department of Electronics & Information Engineering, HUST. His current research areas include computer graphics, multimedia information processing, and computer vision. He is a member of IEEE System, Man & Cybernetics Society.