Query-based learning of acyclic conditional preference networks from noisy data

Fabien Labernia, Florian Yger, Brice Mayag, Jamal Atif

▶ **To cite this version:**

Fabien Labernia, Florian Yger, Brice Mayag, Jamal Atif. Query-based learning of acyclic conditional preference networks from noisy data. EURO Mini Conference: "From Multiple Criteria Decision Aid to Preference Learning" (DA2PL'2016), Nov 2016, Paderborn, Germany. pp.6. hal-01461579

HAL Id: hal-01461579
https://hal.science/hal-01461579

Submitted on 8 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Query-based learning of acyclic conditional preference networks from noisy data

Fabien Labernia and **Florian Yger** and **Brice Mayag** and **Jamal Atif** [1]

Abstract. Conditional preference networks (CP-nets) provide a powerful, compact, and intuitive graphical tool to represent the preferences of a user. However learning such a structure is known to be a difficult problem due to its combinatorial nature. We propose in this paper a new, efficient, and robust query-based learning algorithm for acyclic CP-nets. In particular, our algorithm takes into account the incoherences in the user's preferences or in noisy data by searching in a principled way the variables that condition the other ones. We provide complexity results of the algorithm, and demonstrate its efficiency through an empirical evaluation on synthetic and on real datasets.

1 Introduction

Representing, learning and reasoning over users preferences is a central question in many Artificial Intelligence related fields, and beyond. A large body of research has focused on preference representation and reasoning (e.g. [6, 11, 14, 21, 22, 23]), but few works have concerned their learning (e.g. [7, 8, 9, 12, 15, 16, 18]). This work focuses on **learning combinatorial preferences**, in the framework of conditional preference networks (CP-nets) as introduced in [5].

CP-nets are a formal framework for preference representation based on the notion of **ceteris paribus** (i.e "all other things being equal"). This notion of *ceteris paribus* captures an intuitive idea: it is difficult to express one preference between two totally different objects[2]. Nevertheless, it is easier to express one preference between two almost identical ones. In this work, we consider that *ceteris paribus* means that two objects differ only by one attribute value. For instance, if two hotels differ by their price *ceteris paribus*, it is probably easier to chose one of them. CP-nets implement this notion by factorizing the preferences, leading to a compact graphical representation.

Learning CP-nets is known to be NP-Complete [1, 5, 7], even for acyclic ones. Despite this 'negative' result, some works have tackled this problem, e.g. regression-based learning [9, 10, 17], learning by reduction to 2-SAT [8], and learning by user queries [7, 13, 15].

In all these approaches, the preferences of the users are considered to be coherent ones. Some other works take into account of noisy preferences [18, 19]. A **noise**, also called **incoherence** or **inconsistency**, is a preference that does not correspond to the true one, i.e. "I prefer $\mathbf{a}$ than $\mathbf{b}$" whereas the true preference is "I prefer $\mathbf{b}$ than $\mathbf{a}$". This noise can appear at random, as explained in [4], instead of an adversarial noise [3] which is not studied here.

[1] Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, 75016 PARIS, email: {fabien.labernia,brice.mayag,florian.yger,jamal.atif}@lamsade.dauphine.fr

[2] An object can be a hotel having as a set of attributes: the number of rooms, the price, etc.

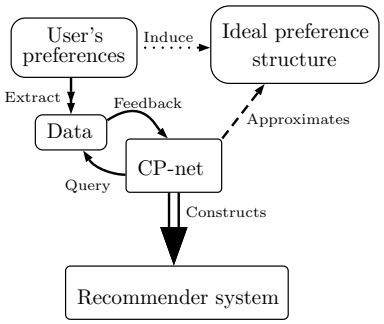

Figure 1. General scheme of a learning procedure for CP-nets.

In this work, we propose a new, efficient, and robust learning algorithm for CP-nets. Our aim is to come up with a learning procedure with the aim of recommendation systems as illustrated in Figure 1. Classically, a CP-net is constructed for each user. However, in this work, we use a CP-net as a way to aggregate preferences of many users. Our learning procedure can be seen as a way to learn an average CP-net of the common preferences between different users. To do this, we exploit the notion of **exact learning**, initially proposed by [2] in the context of query learning.

We focus on the **query learning** paradigm for CP-nets as introduced in [15]. More precisely, we proceed by querying (a set of users or a database) the preference between two objects that differ by just one attribute value), and in case of incoherence detection, we minimize its influence by choosing the optimal attributes that maximize the coherence of the overall learnt CP-net.

Our learning algorithm is composed of two phases: a general learning phase aiming at adding the preference in the graphical structure of the CP-net, and a parent search phase aiming at updating this graphical structure. This two-phases decomposition is classical in CP-nets learning. However, the originality of our approach lies in these both phases. In the general learning phase, our method add the less possible number of preferences in the structure which significantly decreases the computation time, and in the search parent phase, we propose a principle updating strategy that minimizes the incoherence of the overall structure by choosing the parent variable which splits the large number of rules stemming from the data.

2 Preliminaries

Let us first introduce some notations and concepts related to CP-nets [5].

2.1 Conditional preference networks (CP-nets)

Let $\mathbf{V} = \{X_1, \ldots, X_n\}$ be a set of n binary **variables** (variables are denoted by capital letters X and sets of variables are denoted by bold letters $\mathbf{X}$). Each variable $X \in \mathbf{V}$ is associated with a **domain** $Dom(\{X\}) = \{x, x'\}$ (to simplify the notation, we will omit the brackets and write $Dom(X)$) of values it can take. The value $x \in Dom(X)$ of a variable $X \in \mathbf{V}$ is called an **assignment**. We denote by $Dom(\mathbf{V}) = Dom(X_1) \times \ldots \times Dom(X_n)$ the domain of the values of $\mathbf{V}$. We call a **state** vector $\mathbf{x} \in Dom(\mathbf{X})$ which is an assignment of all $X_i \in \mathbf{X}$, with $\mathbf{X} \subseteq \mathbf{V}$ (if $\mathbf{X} = \mathbf{V}$, such a vector is called **outcome**).

We consider in this study a **strict preference relation** as a partial ordering $\succ$ on $Dom(\mathbf{V})$, i.e. $x \succ y$ meaning that an object x **is strictly preferred to** an object y.

Let $\mathbf{x} \in Dom(\mathbf{X})$ and $\mathbf{y} \in Dom(\mathbf{Y})$ be two states, with $\mathbf{X}, \mathbf{Y} \subseteq \mathbf{V}$, $\mathbf{X} \cap \mathbf{Y} = \emptyset$. The notation $\mathbf{xy} \in Dom(\mathbf{X} \cup \mathbf{Y})$ is the concatenation of the state $\mathbf{x}$ and the state $\mathbf{y}$.

Let $\mathbf{o}, \mathbf{o'} \in Dom(\mathbf{V})$ be two outcomes. We call a **swap**, denoted by $(\mathbf{o}, \mathbf{o'})_V$ (which induces $\mathbf{o} \succ \mathbf{o'}$), a pair of two outcomes such that the assignment of only one variable V changes between both. This variable is called the **swap variable** of $(\mathbf{o}, \mathbf{o'})_V$. Furthermore, $\mathbf{o}[\mathbf{X}]$ represents the vector of assignments of all variables in the set $\mathbf{X} \subseteq \mathbf{V}$ (when $\mathbf{X} = \mathbf{V}$, $\mathbf{o}[\mathbf{X}] = \mathbf{o}$).

A variable X is called a **parent variable** of another variable V if the assignment of X changes the preference over the assignments of V. More generally, $Pa(V)$ denotes the set of all parent variables of V. This defines a rule r, called **CP-rule**, of the form $r = (\mathbf{u} : v \succ v')$ with $V \in \mathbf{V}, Dom(V) = \{v, v'\}, \mathbf{U} = Pa(V) \subseteq \mathbf{V} \setminus \{V\}$, and $\mathbf{u} \in Dom(\mathbf{U})$. If $Pa(V) = \emptyset$, we just write $r = (v \succ v')$. We say that a CP-rule $r = (\mathbf{u} : v \succ v')$ **is linked by** V. These rules are stored in a structure called **CP-table** (for conditional preference table), which is unique for each variable $Y \in \mathbf{V}$ and is denoted by $CPT(Y)$. A CP-table contains the preferences over the assignment of Y for some states $\mathbf{x} \in Dom(Pa(Y))$. When all the possible states of $Pa(Y)$ are present, it is said to be **complete**. We note by $CPT(\mathbf{V})$ the union of all CP-tables in $\mathbf{V}$. More formally $CPT(\mathbf{V}) = \bigcup_{V \in \mathbf{V}} CPT(V)$.

Definition 1. *A **conditional preference network** (CP-net) $\mathcal{N} = (\mathbf{V}, A, CPT(\mathbf{V}))$ is a directed graph with $\mathbf{V}$ the set of vertices (representing the variables), A the set of directed arcs such that $(X, Y) \in A$ iff $X \in Pa(Y)$, and $CPT(\mathbf{V}) = \bigcup_{V \in \mathbf{V}} CPT(V)$. A CP-net is said **complete** if its associated CP-tables are complete. We call **separable CP-net** a CP-net $\mathcal{N}$ such that $A = \emptyset$, i.e. $Pa(V) = \emptyset$, $\forall V \in \mathbf{V}$.*

A graphic representation of a CP-net and the partial ordering of all possible outcomes is given in Figure 2. One can note that it is a complete and a non-separable CP-net which contains an independent variable (swimming pool) and a variable conditioned by another one (kitchen is conditioned by occupancy).

We say that a CP-net $\mathcal{N}$ is **cyclic** iff there exists a cycle in its associated graph. Otherwise, the CP-net is said **acyclic**. We restrict our work to acyclic CP-nets.

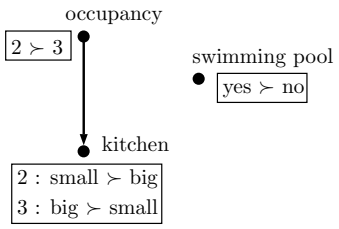

Figure 2. Complete CP-net with three variables.

2.2 Query learning algorithms for CP-nets

We discuss in this section the main query learning algorithms for CP-nets. We denote by n the number of variables of a CP-net, i.e. $|\mathbf{V}| = n$, by $\mathcal{N}_T$ the target CP-net, i.e. the structure that we want to fit, and by $\mathcal{N}_L$ our learned CP-net.

The first reported query learning algorithm for CP-nets in the literature [15] proceeds by asking different questions to a user in order to learn her preferences. The working assumption is that the user preferences are representable by a CP-net structure. Two types of questions can be distinguished:

- **equivalence queries** $\texttt{EQ}(\mathcal{N}_L, \mathcal{N}_T)$ which return `TRUE` if $\mathcal{N}_L$ is equivalent to $\mathcal{N}_T$[3], otherwise they return `FALSE` plus a counterexample in a form of a swap $(\mathbf{o}, \mathbf{o'})_V$ s.t. $\mathbf{o} \succ \mathbf{o'}$ representing a violated rule r.
- **membership queries** $\texttt{MQ}(\mathcal{N}_T, r)$, which return `TRUE` if the rule r is satisfied in $\mathcal{N}_T$ (denoted by $\mathcal{N}_T \models r$), otherwise they return `FALSE`.

When $\mathcal{N}_L \neq \mathcal{N}_T$, the equivalence query returns a swap counterexample $(\mathbf{o}, \mathbf{o'})_V$. This swap induces a rule r which (1) if it does not exist in $\mathcal{N}_L$, then we just have to add r in $\mathcal{N}_L$, or (2) if it is violated in $\mathcal{N}_L$, then we have to find a new parent variable for V. In [15], it has been shown that a linear number of equivalence queries ($O(|\mathbf{V}|)$) and a logarithmic number of membership queries ($O(\log_2 |\mathbf{V}|)$) are required to learn such a CP-net. For an in-depth explanation of the method, the interested reader can refer to [15].

The second algorithm [13] is neither based on *ceteris paribus* comparison nor equivalence and membership queries. It is an **online** algorithm that learns an **acyclic** CP-net and is decomposed into two phases:

- finding a separable CP-net by asking for each variable V the preference between v and v',
- updating the CP-table of each variable by finding the best set of parent variables using a set of confident[4] variables.

In this algorithm, all the parent variables are selected at the same time by phase 2. It seeks a subset of parent variables $\mathbf{P}$ from the set of all confident variables $\mathbf{C}$, i.e. $\mathbf{P} \subseteq \mathbf{C} \subseteq \mathbf{V}$. Moreover, for all $\mathbf{P} \subseteq \mathbf{C}$, the entire CP-table of the current variable V is created and tested until a good one is found (i.e. a CP-table that satisfies all the preferences). In the worst case, this algorithm needs 2^{2^n} operations to determine, for each variable, its parents and its corresponding CP-table.

[3] We say that two CP-nets $\mathcal{N}$ and $\mathcal{N}'$ are equivalent, denoted by $\mathcal{N} \equiv \mathcal{N}'$ iff they induce exactly the same preferences.

[4] We consider a variable V as **confident** if enough swap that induced rules of V are found.

Due to the exponential nature of this phase, it is necessary to limit the computation by bounding the size p of parent variables, the number e of edges in the target CP-net $\mathcal{N}_T$, and the number q of necessary swaps to conclude that a variable becomes confident. Finally, the algorithm can learn a CP-net in $O(n^p)$, with p the maximum number of parent variables. For more details, we refer the reader to [13].

3 Proposed algorithm for learning a CP-net

Let $r = (\mathbf{u} : v \succ v')$ be a rule with $V \in \mathbf{V}, Dom(V) = \{v, v'\}, Pa(V) = \mathbf{U} \subseteq \mathbf{V} \setminus \{V\}$ and $\mathbf{u} \in Dom(\mathbf{U})$. For the sake of clarity, we introduce in this section the notations $\bar{r} = (\mathbf{u} : v' \succ v)$ as the inverse rule of V, and $r^p = (\mathbf{u}p : v \succ v')$ as the augmentation of rule r with an assignment $p \in Dom(P)$ of a new parent variable $P \in \mathbf{V} \setminus (\mathbf{U} \cup \{V\})$.

As in the algorithm in [15], we query an oracle through $\texttt{EQ}(\mathcal{N}_L, \mathcal{N}_T)$ if our learned CP-net $\mathcal{N}_L$ is equivalent to its induced (the target) CP-net $\mathcal{N}_T$. This oracle can be either a user or a dataset that returns `TRUE` or `FALSE`. We suppose here that the oracle is able to answer, in a polynomial time, the following three queries:

1. the equivalence between two CP-nets (and returns a counterexample if not),
2. its preference between two outcomes in a swap,
3. another swap $(\mathbf{o}'', \mathbf{o}''')_V$ which respects some conditions (see Eq. (1) in Subsection 3.2).

We denote by Σ the oracle, and by $\mathcal{N}_T$ its proper CP-net. The target CP-net $\mathcal{N}_T$, cannot generally be explicitly known due to e.g. cognitive overloads for a user or the size of the database. However, we suppose that the oracle is able to differentiate its proper CP-net from the learned one.

Our algorithm, contrary to the state of the art approaches, tries to take into account incoherent (contradictory) preferences. Indeed, it may happen, for a user, to have **incoherent** preferences (she makes some mistakes or her preferences cannot be modeled by a CP-net). In case of databases, the data can be affected by noise, i.e. contradictory preferences. Hence, the learning procedure should be robust as much as possible to such a noise. In our approach, we introduce a **list of violated rules** L which cannot be represented in our CP-net either because we cannot add a parent to a variable to represent the rule, or even if such a parent exists, this rule cannot hold in $\mathcal{N}_L$. Then, we say that two CP-nets $\mathcal{N}_L$ and $\mathcal{N}_T$ are equivalent if the oracle compares these two CP-nets without using the rules contained in L, i.e. $\texttt{EQ}(\mathcal{N}_L, \Sigma, L)$.

Following previous learning algorithms, we decompose our procedure into a general learning phase, and a parent search phase. Still, our two phases are different from the ones in the state of the art.

3.1 General learning phase

In exact learning theory, we look for a perfect equivalence between the target and the learned structure. Following this perspective, we need to completely fit $\mathcal{N}_L$ and improve as much as possible the learning accuracy. Algorithm 1 corresponds to the general learning phase, starting with an empty CP-net $\mathcal{N}_L$.

$\mathcal{N}_L$ is updated by the rule r induced by the swap counterexample provided by the oracle. As in [5], two cases can occur: the inverse rule $\bar{r}$ is not present in $\mathcal{N}_L$, then we just have to add r to our CP-net. But if $\bar{r}$ is already present, then we must find a new parent to the variable V associated with the rule r. Since our Algorithm 1 does not always pick the real good parent (because of the presence of incoherences in the data), the parent search procedure can fail. If this happens (Line 11), we add r in the list of violated rules (i.e. a list that cannot be represented in $\mathcal{N}_L$). Otherwise (Line 7), we remove all the $CPT(V)$ and create a new one that contains r^p and $\bar{r}^{p'}$.

Algorithm 1: $\texttt{learningCPNet}(\Sigma, \mathcal{N}_T)$

Data: An oracle Σ which induces a target CP-net $\mathcal{N}_T$.
Result: A learned CP-net $\mathcal{N}_L$.

1 $L = \emptyset$;
2 $\mathcal{N}_L = (\mathbf{V}, A = \emptyset, CPT(\mathbf{V}) = \emptyset)$;
3 **while** *($\neg\texttt{EQ}(\mathcal{N}_L, \mathcal{N}_T, L)$)* **do**
4 Let $(\mathbf{o}, \mathbf{o}')_V \in \Sigma$ be a swap counterexample returned by `EQ` and $r = (\mathbf{u} : v \succ v')$ its induced rule with $\mathbf{U} = Pa(V)$;
5 **if** *($\bar{r} \in CPT(V)$)* **then**
6 $P \leftarrow \texttt{searchParent}((\mathbf{o}, \mathbf{o}')_V, \Sigma)$;
7 **if** *(P exists, with $Dom(P) = \{p, p'\}$)* **then**
8 $A \leftarrow A \cup \{(P, V)\}$;
9 $CPT(V) \leftarrow \{r^p, \bar{r}^{p'}\}$ where $p = \mathbf{o}[P]$;
10 $L \leftarrow L \setminus \{r'\},\ \forall r' \in L$ s.t. r' is linked by V;
11 **else** $L \leftarrow L \cup \{r\}$;
12 **else** $CPT(V) \leftarrow CPT(V) \cup \{r\}$;
13 **return** $\mathcal{N}_L$;

The algorithms in [13, 15] try to compute the complete CP-table of V and restart for each new parent variable, which leads to heavy computations. Nevertheless, we guess that in real world applications, CP-tables are not generally complete. Moreover, in case when the oracle refines its judgment, the preference in a swap can be different from a moment to the other, and an incoherence becomes coherent. Unfortunately, the opposite is also possible. However, we expect that its preferences becomes truthful as the time passes. Then it is not mandatory to compute the complete one in a greedy manner. This allows us to reduce the computational time. Besides, the missing entries are detected in our algorithm thanks to the equivalence queries.

3.2 Parent search phase

The parent search phase is the most important one in the learning procedure. This is due to the fact that several parent variables can be chosen, which may lead to bad decisions.

The procedure `searchParent` needs as an input the swap counterexample given by `EQ` and the oracle Σ. Its first step is to query the parent variable P of the swap $(\mathbf{o}, \mathbf{o}')_V$ (with $\mathbf{o} \succ \mathbf{o}'$) associated with variable V. The variable P must satisfy the following conditions:

1. It should preserve the assignment p between two comparable[5] outcomes. This is trivial in our case because we have a swap.
2. There should exist at least one other swap $(\mathbf{o}'', \mathbf{o}''')_V$ associated with the same variable V such that the preference on V is reversed and it contains the inverse assignment of P.

We can summarize these conditions in the following equation: let $(\mathbf{o}, \mathbf{o}')_V$ and $(\mathbf{o}'', \mathbf{o}''')_V$ two swaps, with $V, P \in \mathbf{V}$, which respect

$$\begin{aligned} &\mathbf{o}[V] = \mathbf{o}'''[V] \neq \mathbf{o}'[V] = \mathbf{o}''[V], \\ \text{and}\quad &\mathbf{o}[P] = \mathbf{o}'[P] \neq \mathbf{o}''[P] = \mathbf{o}'''[P]. \end{aligned} \tag{1}$$

[5] Two outcomes $\mathbf{o}$ and $\mathbf{o}'$ are **comparable** if either $\mathbf{o} \succ \mathbf{o}'$ or $\mathbf{o}' \succ \mathbf{o}$.

These constraints are modeled in Line 1. Since we restrict ourselves to acyclic CP-nets, a function `cycle` is used to test the acyclicity of $\mathcal{N}_L$ with the new parent variable.

We need to choose the good parent variable among all the available ones in $\mathbf{P}$. Instead of choosing a random $P \in \mathbf{P}$, we pick the variable P that minimizes the number of swaps that violate the rule induced by the current swap counterexample, i.e. let $(\mathbf{o}, \mathbf{o}')_V$ and $(\mathbf{o}'', \mathbf{o}''')_V$ be two swaps, with $V \in \mathbf{V}$ and $P \in Pa(V)$, then:

$$\begin{array}{rl} & (\mathbf{o}[P] = \mathbf{o}''[P] \text{ and } \mathbf{o}[V] \neq \mathbf{o}''[V]) \\ \text{or} & (\mathbf{o}[P] \neq \mathbf{o}''[P] \text{ and } \mathbf{o}[V] = \mathbf{o}''[V]). \end{array} \quad (2)$$

In case of equality, we randomly choose one.

Our parent search procedure is given in Algorithm 2. It is important to note that this algorithm can be parallelized using one process for each parent candidate, thanks to the independence of this search. However, this implementation is left for future work.

Algorithm 2: `searchParent`$((\mathbf{o}, \mathbf{o}')_V, \Sigma)$

Data: A swap $(\mathbf{o}, \mathbf{o}')_V$ and the oracle Σ.
Result: A parent variable P if there exists one, an error otherwise.

1 $\mathbf{P} \leftarrow \{P \in \mathbf{V} \setminus (\{V\} \cup Pa(V)) \mid \neg\texttt{cycle}(\mathcal{N} = (\mathbf{V}, A \cup \{(P, V)\}, CPT(\mathbf{V}))) \text{ and } \exists (\mathbf{o}'', \mathbf{o}''')_V \in \Sigma \text{ s.t. Eq. (1) returns } \texttt{TRUE}\}$;
2 **if** $(\mathbf{P} \neq \emptyset)$ **then**
3 $\quad$ **return** $\underset{P \in \mathbf{P}}{\operatorname{argmin}} \{\#(\mathbf{o}'', \mathbf{o}''')_V \in \Sigma \mid \text{Eq. (2) returns } \texttt{TRUE}\}$;
4 **else return** "parent not found";

Example 1. *Let us now try to learn a simple complete CP-net from Figure 2 (without the swimming pool variable). We begin by asking a couple if the empty learned CP-net is equivalent to their induced one. This is obviously not the case. Consider that they then give us the following swap counterexample* $((2,\ big), (3,\ big))_{occupancy}$*. The algorithm finds the rule* $r = (2 \succ 3)$ *which is added to* $\mathcal{N}_L$*. The next step is the equivalence query. Consider now that they answer the query by returning* $((2,\ small), (2,\ big))_{kitchen}$*. Then, the rule* $r = (small \succ big)$ *is deduced, and is added to* $\mathcal{N}_L$*. The process is repeated once again. Consider once more that the couple at this stage returns the following swap counterexample* $((3,\ big), (3,\ small))_{kitchen}$ *which induces the rule* $r = (big \succ small)$*. Since the inverse rule* $\bar{r} = (small \succ big)$ *already exists, the* `searchParent` *returns only the occupancy variable. Theses two new rules* $r^3 = (3 : big \succ small)$ *and* $\bar{r}^2 = (2 : small \succ big)$ *are then added in the CP-net.*

We end this subsection by giving some complexity results about these two algorithms.

Proposition 1. *Let* Σ *be an oracle. We define by* s *the number of swaps contained in* Σ*, i.e. holding in a database, or known by a user, and by* n *the number of variables in a CP-net. Algorithm 2 has a complexity of* $O(n^3 + ns)$*.*

Proof : Line 1 has to detect a cycle in $O(n+n^2) \approx O(n^2)$. The first n is the number of variables in the CP-net and the second n^2 corresponds to the maximum number of directed arcs in a directed graph. Finding a swap that respects the parent condition can be computed in $O(s)$. These steps have to be repeated for each parent candidate. Thus, Line 1 has a total complexity of $O(n(s + n^2) = O(n^3 + ns)$. Line 3 has to count the number of swaps in Σ that respect a given condition, it is in $O(ns)$. Hence, Algorithm 2 has a total complexity of $O(n^3 + ns)$. ■

Proposition 2. *Algorithm 1 has a complexity of* $O(2^p(n^4 + n^2 s + ne))$ *to compute* $\mathcal{N}_L$*, where* $p = \max_{V \in \mathbf{V}} \{|Pa(V)|\}$*,* e *is the time taken by* `EQ` *to return* `TRUE` *or* `FALSE`*, and* $\mathcal{N}_L$ *is an acyclic CP-net.*

Proof : We know from Prop. 1 that `searchParent` (Line 6) has a complexity of $O(n^3+ns)$. We now consider the `while` condition at Line 3. Suppose that $\mathcal{N}_T$ is a complete CP-net. Then, we must have complete CP-tables which imply, for p the max number of parents in $\mathcal{N}_T$, 2^p equivalence queries (one for each rule). Moreover, we have to remove all the rules in the CP-table when a new parent is found. In the worst case, we need $\sum_{i=1}^{p} 2^i = 2(2^p - 1)$ equivalence queries to learn one CP-table, so $2n(2^p - 1)$ equivalence queries in total. We finally have a complexity of $2n(2^p - 1)(n^3 + ns + e) \in O(2^p(n^4 + n^2 s + ne))$. ■

Note that s and e, which are respectively defined in Propositions 1 and 2, correspond to the same process if the oracle is a dataset. For n the number of variables, the maximum number of objects in a dataset is 2^n in the binary case. The maximum number of rules is then $k \leq 2^n$. Furthermore, each of these rules is represented by at least one swap (exactly one if $k = 2^n$). Hence, in the worst case, $s \leq 2^n$ and $e \leq 2^n$. However, in real world applications, $s < m \ll 2^n$ (with m the number of objects in a dataset) and the worst case does not hold in most of situations.

4 Experimental results

In this section, we consider that the oracle Σ is a database that contains a list of swaps. To evaluate the efficiency of our algorithm, we demonstrate its usefulness on real and on simulated datasets.

Defining an accuracy measure of a learning algorithm on such a structure is not a trivial task. A CP-net can be seen as a set of rules. A workaround to define an accuracy measure is to use the number of rules induced by the preferences of Σ which are correctly represented in $\mathcal{N}_L$ versus the total number of rules. However, even if this can be relevant in some cases, where there is just one violated rule, this measure becomes meaningless. Indeed, a violated rule cannot be represented in $\mathcal{N}_L$ and then induces a huge number of unsatisfiable swaps. In a context of learning a relevant CP-net, we rather prefer to use the number of swaps that are in agreement with $\mathcal{N}_L$ versus the total number of swaps in the whole dataset.

We use two different runs of our algorithm in order to smooth our results: the first one consists of the random generation phase of the target CP-nets or the datasets, and the second one consists of the learning phase of $\mathcal{N}_L$. We note these two runs by $k \times l$ with k the random generation phase and l the learning phase. In all the graphics in the rest of the paper, each point corresponds to a simple averaged value according to the number of runs. The standard deviation is reported as an error bar on these graphics.

We use two distinct datasets to conduct our experiments:

1. the TripAdvisor dataset [24, 25] rescaled to obtain binary attributes (considered here as variables),
2. randomly generated dataset in order to test the scalability of our algorithm, and its robustness to incoherences.

The TripAdisor[6] dataset contains about $240,000$ hotel reviews. A hotel is represented by seven rates (between 1 and 5) plus one general rate. We use this general rate as our preference relation between the reviews. To be able to learn a binary CP-net, the rates are rescaled: `1` if the rate is strictly greater than two, and `0` otherwise. We have, after this procedure, 126 different hotel reviews that induce a target CP-net $\mathcal{N}_T$ which potentially contains incoherences.

We also generate a random artificial dataset as follows: two random boolean vectors such that they form a swap (only one bit changes between both vectors) are generated along with a random score for each of these vectors. This score corresponds to our preference relation. In order to test Algorithm 1, we generate three sizes of random datasets with 50 objects (7 attributes), 500 objects (10 attributes), and 10000 objects (15 attributes). We set the size of vectors of each dataset by applying $n = \lfloor \log_2 m \rfloor + 1$, with m the number of objects in the dataset. We suppose that such a generated synthetic dataset reflects reality in the sense that a real dataset cannot contain all the possible combinations of the attribute values and some objects may not exist, i.e. $m \leq 2^{n-1} < 2^n$. A preprocessing procedure transforms a set of objects into a set of swaps.

Accuracy(%)	Agreement	Disagreement
	$p = 1$	
real hotels (126)	0.73	0.27
random hotels (50)	0.72	0.28
random hotels (500)	0.57	0.43
random hotels (10000)	0.52	0.48
	$p = 5$	
real hotels (126)	0.82	0.18
random hotels (50)	0.82	0.18
random hotels (500)	0.59	0.41
random hotels (10000)	0.51	0.49
	$p = \infty$	
real hotels (126)	0.83	0.17
random hotels (50)	0.79	0.21
random hotels (500)	0.69	0.31
random hotels (10000)	0.66	0.34
	relaxing acyclic condition	
real hotels (126)	1.00	0.00
random hotels (50)	1.00	0.00
random hotels (500)	1.00	0.00
random hotels (10000)	*too long to compute*	

Table 1. Accuracy of Algorithm 1. Results are averaged on 10×10 runs.

We firstly test the importance of having coherent preferences for the learning phase. Except for the 50 objects dataset (probably due to the few number of objects), a gap is observed between the real hotel dataset and the 500 objects dataset in Figure 3. Of course, the agreement grows with regards to the maximum number of parents. The last test consists in relaxing the acyclic condition in order to observe the its influence. In this case, we are able to exactly fit $\mathcal{N}_T$, but at the price of heavy computations.

Figure 3 depicts the results of Table 1. The accuracy increases linearly w.r.t the number of parents. Once again, one can observe a gap between the accuracy of separable ($p = 0$) and tree-shaped CP-nets ($p = 1$) for all datasets, and a decreasing accuracy between $p = 1$ and $p = 2$. The accuracy continues growing when $p > 2$. Furthermore, one can note that the learning procedure is not stable

[6] `http://times.cs.uiuc.edu/~wang296/Data/`.

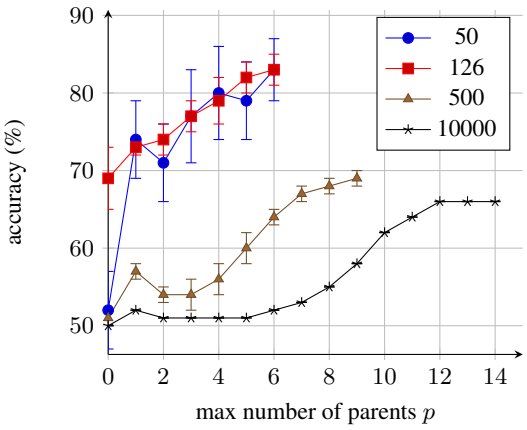

Figure 3. Learning accuracy according to the number of parent p per variable. Datasets are randomly generated except for the real 126 hotels file. Results are averaged on 5×10 runs and error bars correspond to the standard deviation of the observed values.

for small numbers of objects ($n < 500$) because of the number of variables which does not allow the algorithm to correct its mistakes.

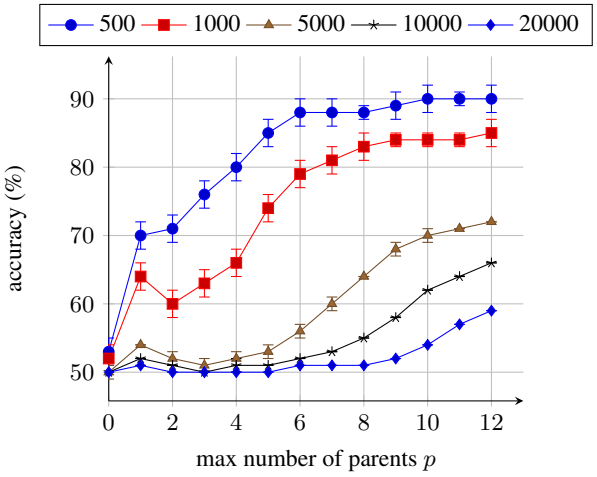

Figure 4. Learning accuracy when the number of variables is fixed ($n = 15$). We increase the number of objects in the random datasets. Results are averaged on 2×5 runs and error bars correspond to the standard deviation of the observed values.

Figure 4 shows the influence of the number of objects m on the accuracy for a fixed number of variables n. When $m \ll 2^n$, we can easily fit the structure with an accuracy greater than 80% (for the 500 and 1000 datasets). When the dataset contains a number of objects m close to the limited size 2^n, the accuracy increases until about 60%. We can observe the same phenomenon as in Figure 3, with a negative gap between $p = 1$ and $p = 2$. It occurs for all random datasets.

At last, we focus on the time taken by Algorithm 1 to learn one CP-net from a dataset. We can see in Figure 5 that this learning task is immediate for $m \leq 1000$. However, whereas we need about 20 seconds to learn a CP-net from $m = 5000$ and $p = 14$, we need more than 150 seconds to learn a CP-net from $m = 10000$ and $p = 14$. Thus, for two times more objects, we need ten times longer to com-

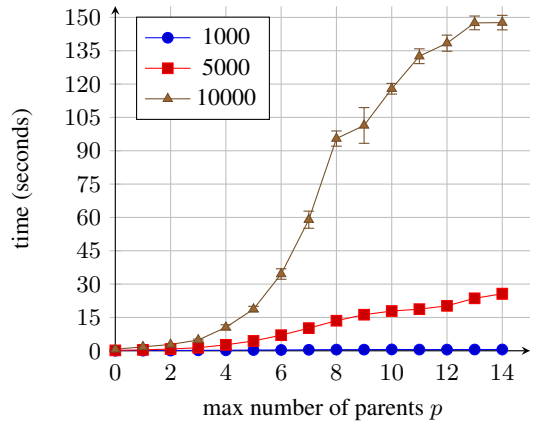

Figure 5. Learning time when the number of variables is fixed ($n = 15$). We increase the number of objects in the datasets. Results correspond to one learning execution averaged on 2×5 runs and error bars correspond to the standard deviation of the observed values.

pute it. However, the computation time increases linearly according to the number of parents: as we need to browse the whole database, the number of objects m (thus the number of swaps s that induce the rules) is a critical factor.

5 Conclusion

We presented in this paper a new algorithm for learning acyclic CP-nets, and designed a bunch of experiments to evaluate its performances on synthetic and on real datasets. They showed that despite its exponential complexity depending on the number of objects and parents (Proposition 2), our algorithm can learn in a few seconds a CP-net on random CP-nets, random datasets, or real dataset.

The main ingredient of our approach is incoherence handling in the preferences. The robustness of our algorithm to these incoherences has been evidenced by the designed experiments. For instance, on a task for hotel rating using a TripAdvisor dataset affected by noise, our algorithm achieves good accuracy results.

Future work will concern the improvement of our algorithm from different standpoints, in particular to reduce its time complexity.

A first effort will concern the implementation of a parallel version of our `searchParent` subroutine in order to decrease the learning time according to Proposition 1. This will allow to gain at least a n factor. This is especially important as recommender systems are applied in environments with massive datasets such as social networks.

A second effort will concern the improvement of several critical parts of our algorithm. The first one concerns the exhaustive nature of equivalence queries. Since we want a perfect fitting between $\mathcal{N}_T$ and $\mathcal{N}_L$ (when working with datasets) we must look over all the input data for an answer, which is very time consuming. This issue can be overcome by designing an approximate strategy, potentially with accuracy guarantees. The second critical part concerns the random nature of the counterexample returned by the equivalence queries. It may occur when the counterexample does not correspond to the real true preference rule. This leads to erroneous rules generation.

REFERENCES

[1] Eisa Alanazi, Malek Mouhoub, and Sandra Zilles, 'The complexity of learning acyclic cp-nets', in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 1361–1367, (2016).

[2] Dana Angluin, 'Queries and concept learning', *Machine learning*, **2**(4), 319–342, (1988).

[3] Dana Angluin, Mārtiņš Kriķis, Robert H Sloan, and György Turán, 'Malicious omissions and errors in answers to membership queries', *Machine Learning*, **28**(2-3), 211–255, (1997).

[4] Dana Angluin and Donna K Slonim, 'Randomly fallible teachers: Learning monotone dnf with an incomplete membership oracle', *Machine Learning*, **14**(1), 7–26, (1994).

[5] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole, 'Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements', *J. Artif. Intell. Res.(JAIR)*, **21**, 135–191, (2004).

[6] Ronald J Brachman, Hector J Levesque, and Raymond Reiter, *Knowledge representation*, MIT press, 1992.

[7] Yann Chevaleyre, Frédéric Koriche, Jérôme Lang, Jérôme Mengin, and Bruno Zanuttini, 'Learning ordinal preferences on multiattribute domains: The case of cp-nets', in *Preference learning*, 273–296, Springer, (2011).

[8] Yannis Dimopoulos, Loizos Michael, and Fani Athienitou, 'Ceteris paribus preference elicitation with predictive guarantees.', in *IJCAI*, volume 9, pp. 1–6. Citeseer, (2009).

[9] Alan Eckhardt and Peter Vojtás, 'How to learn fuzzy user preferences with variable objectives.', in *IFSA/EUSFLAT Conf.*, pp. 938–943, (2009).

[10] Alan Eckhardt and Peter Vojtáš, 'Learning user preferences for 2cp-regression for a recommender system', in *SOFSEM 2010: Theory and Practice of Computer Science*, 346–357, Springer, (2010).

[11] Peter C Fishburn, *Decision and value theory*, number 10, Wiley New York, 1964.

[12] Johannes Fürnkranz and Eyke Hüllermeier, *Preference learning*, Springer, 2011.

[13] Joshua T Guerin, Thomas E Allen, and Judy Goldsmith, 'Learning cp-net preferences online from user queries', in *Algorithmic Decision Theory*, 208–220, Springer, (2013).

[14] Ralph L Keeney and Howard Raiffa, *Decisions with multiple objectives: preferences and value trade-offs*, Cambridge university press, 1993.

[15] Frédéric Koriche and Bruno Zanuttini, 'Learning conditional preference networks', *Artificial Intelligence*, **174**(11), 685–703, (2010).

[16] Jérôme Lang and Jérôme Mengin, 'Learning preference relations over combinatorial domains', *Proceedings of NMR08*, 207–214, (2008).

[17] Juntao Liu, Chenhong Sui, Dewei Deng, Junwei Wang, Bin Feng, Wenyu Liu, and Caihua Wu, 'Representing conditional preference by boosted regression trees for recommendation', *Information Sciences*, **327**, 1–20, (2016).

[18] Juntao Liu, Yi Xiong, Caihua Wu, Zhijun Yao, and Wenyu Liu, 'Learning conditional preference networks from inconsistent examples', *Knowledge and Data Engineering, IEEE Transactions on*, **26**(2), 376–390, (2014).

[19] Juntao Liu, Zhijun Yao, Yi Xiong, Wenyu Liu, and Caihua Wu, 'Learning conditional preference network from noisy samples using hypothesis testing', *Knowledge-Based Systems*, **40**, 7–16, (2013).

[20] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell, *Machine learning: An artificial intelligence approach*, Springer Science & Business Media, 2013.

[21] Tuomas Sandholm, 'Expressive commerce and its application to sourcing: How we conducted $35 billion of generalized combinatorial auctions', *AI Magazine*, **28**(3), 45, (2007).

[22] Alexis Tsoukiàs, 'From decision theory to decision aiding methodology', *European Journal of Operational Research*, **187**(1), 138–161, (2008).

[23] Toby Walsh, 'Representing and reasoning with preferences', *AI Magazine*, **28**(4), 59, (2007).

[24] Hongning Wang, Yue Lu, and Chengxiang Zhai, 'Latent aspect rating analysis on review text data: a rating regression approach', in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 783–792. ACM, (2010).

[25] Hongning Wang, Yue Lu, and ChengXiang Zhai, 'Latent aspect rating analysis without aspect keyword supervision', in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 618–626. ACM, (2011).