EURO J Decis Process
DOI 10.1007/s40070-017-0070-3

ORIGINAL ARTICLE

Query-based learning of acyclic conditional preference networks from contradictory preferences

Fabien Labernia[1] · Florian Yger[1] · Brice Mayag[1] · Jamal Atif[1]

Received: 10 February 2017 / Accepted: 6 September 2017

Abstract Conditional preference networks (CP-nets) provide a 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 contradictions between multiple users' preferences by searching in a principled way the variables that affect the preferences. We provide complexity results of the algorithm, and demonstrate its efficiency through an empirical evaluation on synthetic and on real databases.

Keywords Query-based learning algorithm · Contradictory preferences · Preference learning · Conditional preference networks

1 Introduction

Representing, learning, and reasoning over users preferences are a central question in many Artificial Intelligence-related fields. An important body of research has focused on preference representation and reasoning in different ways, e.g.,

✉ Fabien Labernia
fabien.labernia@lamsade.dauphine.fr

Florian Yger
florian.yger@lamsade.dauphine.fr

Brice Mayag
brice.mayag@lamsade.dauphine.fr

Jamal Atif
jamal.atif@lamsade.dauphine.fr

[1] Université Paris-Dauphine, PSL Research University, CNRS, UMR 7243, LAMSADE, Paris 75016, France

Published online: 21 September 2017

conditional preference networks' (CP-nets) representations (Boutilier et al. 2004), and in decision theory for the decision aiding (Tsoukiàs 2008). However, only few works have concerned their learning (e.g., preference learning problems (Fürnkranz and Hüllermeier 2010), or learning CP-nets representation (Chevaleyre et al. 2010; Koriche and Zanuttini 2010; Liu et al. 2014). This work focuses on learning combinatorial preferences, in the framework of CP-nets as introduced in Boutilier et al. (2004).

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,[1] 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 choose 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 (Alanazi et al. 2016; Boutilier et al. 2004; Chevaleyre et al. 2010), even for acyclic ones. Despite this 'negative' result, some works have tackled this problem, e.g., regression-based learning (Eckhardt and Vojtás 2009; Eckhardt and Vojtás 2010; Liu et al. 2016), learning by reduction to 2-SAT (Dimopoulos et al. 2009), and learning by user queries (Chevaleyre et al. 2010; Guerin et al. 2013; Koriche and Zanuttini 2010).

In all approaches cited above, the preferences of the users are considered to be coherent ones (i.e., non-contradictory preferences—some other works (Liu et al. 2013, 2014) manage the contradictions by removing cycles in the preferences).

A contradictory preference is a preference, such that the reversed one also exists. This can be the result of two users that express their true preferences but have different opinions, i.e., in a multiple users database, person A will prefer an alternative **a** to an alternative **b**, whereas person B will prefer **b** to **a**.

In this paper, we consider the problem of learning from users' preferences that may be contradictory. We propose a new, efficient, and robust learning algorithm for CP-nets. Our aim is to come up with a learning procedure applicable to recommendation systems, as illustrated in Fig. 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 Angluin (1987) in the context of query learning as introduced in Koriche and Zanuttini (2010). 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 contradiction detection, we minimize its influence by choosing the optimal attributes that maximize the coherence of the overall learned 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

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

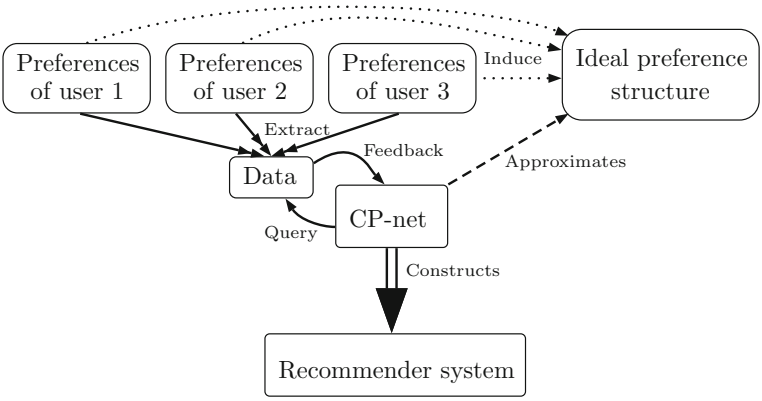

Fig. 1 General scheme of a learning procedure for CP-nets

search phase aiming at updating this graphical structure. This two-phase decomposition is classical in CP-nets learning. However, the originality of our approach lies in both these phases. In the general learning phase, our method adds the less possible number of preferences in the structure which significantly decreases the computation time, and in the search parent phase, we propose a principled updating strategy leading to the minimization of the contradictions in the overall structure by choosing the parent variable that splits the large number of rules stemming from the data.

2 Preliminaries

Let us first introduce some notations and concepts related to CP-nets (Boutilier et al. 2004).

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 $\mathbf{X}$), also called world. 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 $\mathbf{x} \in Dom(\mathbf{X})$ 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).

Example 1 Let us consider a hotel that contains a set of rooms (defined by their available occupancy (2 or 3 places), the size of the kitchen (small or big), and the presence (or not) of a swimming pool). More formally, we consider a world $\mathbf{V}$ containing three variables $\mathbf{V} = \{\text{occupancy, kitchen, swimming pool}\}$, with

$Dom(\text{occupancy}) = \{2, 3\}$, $Dom(\text{kitchen}) = \{\text{small}, \text{big}\}$, and $Dom(\text{swimmingpool}) = \{\text{yes}, \text{no}\}$. Thus, a state (in this case, an outcome) $\mathbf{h}$ is the vector $\mathbf{h} = (2, \text{small}, \text{no})$ standing for a room with 2 places, and a small kitchen without swimming pool.

We consider in this study a strict preference relation as a partial ordering[2] $\succ$ on $Dom(\mathbf{V})$, i.e., $\mathbf{x} \succ \mathbf{y}$ meaning that an outcome $\mathbf{x}$ is strictly preferred to an outcome $\mathbf{y}$.

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 $\mathbf{y}$.

Definition 1 (Preferential independence) Let $\mathbf{V}$ be a world. A set of variables $\mathbf{X} \subseteq \mathbf{V}$ is preferentially independent of its complement $\mathbf{Y} = \mathbf{V} \backslash \mathbf{X}$ iff $\forall \mathbf{x}, \mathbf{x}' \in Dom(\mathbf{X})$ and $\forall \mathbf{y}, \mathbf{y}' \in Dom(\mathbf{Y})$, we have

$$\mathbf{xy} \succ \mathbf{x}'\mathbf{y} \text{ iff } \mathbf{xy}' \succ \mathbf{x}'\mathbf{y}'.$$

The previous definition means that regardless the assignments of other variables, if we prefer $\mathbf{x}$ to $\mathbf{x}'$, this relation always holds.

Example 2 Let us say that regardless the number of persons who want to rent a hotel room, everyone prefers to have a swimming pool rather than nothing. One can justify this statement by the fact that the price of the room is not affected by the presence or not of the swimming pool. Hence, we say that the swimming pool variable is preferentially independent of the occupancy and the kitchen variables.

Definition 2 (Conditional preferential independence) Let $\mathbf{V}$ be a world, and $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$ be nonempty sets that partition $\mathbf{V}$. $\mathbf{X}$ is conditionally preferentially independent of $\mathbf{Y}$ given a state $\mathbf{z} \in Dom(\mathbf{Z})$ iff $\forall \mathbf{x}, \mathbf{x}' \in Dom(\mathbf{X})$ and $\forall \mathbf{y}, \mathbf{y}' \in Dom(\mathbf{Y})$, we have

$$\mathbf{xyz} \succ \mathbf{x}'\mathbf{yz} \text{ iff } \mathbf{xy}'\mathbf{z} \succ \mathbf{x}'\mathbf{y}'\mathbf{z}.$$

If $\mathbf{X}$ is conditionally preferentially independent of $\mathbf{Y}$ for all $\mathbf{z} \in Dom(\mathbf{Z})$, then $\mathbf{X}$ is conditionally preferentially independent of $\mathbf{Y}$ given $\mathbf{Z}$.

Definition 2 means that (1) $\mathbf{X}$ is preferentially independent of $\mathbf{Y}$ *ceteris paribus*, and (2) the preferences over the states of $\mathbf{X}$ change according to the assignments of the set of variables $\mathbf{Z}$. We say that $\mathbf{Z}$ conditions $\mathbf{X}$.

Example 3 Suppose we have a couple (with a child) who needs to rent a hotel room in the previous world $\mathbf{V} = \{\text{occupancy, kitchen, swimming pool}\}$. When they are without their child, they prefer to eat in a restaurant, so having a small kitchen is rational. However, if they are with their child, they prefer to cook, so they need a big kitchen. This shows that the kitchen variable **is conditioned by** the occupancy variable.

[2] A partial ordering $\succ$ is an asymmetric, irreflexive, and transitive relation, i.e., if $\mathbf{x} \succ \mathbf{y}$, then $\mathbf{y} \not\succ \mathbf{x}$ (asymmetry), $\mathbf{x} \not\succ \mathbf{x}$ (irreflexivity), and if $\mathbf{x} \succ \mathbf{y}$ and $\mathbf{y} \succ \mathbf{z}$, then $\mathbf{x} \succ \mathbf{z}$ (transitivity).

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 $\mathbf{X} = \mathbf{V}$, $\mathbf{o}[\mathbf{X}] = \mathbf{o}$).

Example 4 Let $\mathbf{h}_1 = (2, \text{small}, \text{yes}), \mathbf{h}_2 = (3, \text{small}, \text{yes})$, and $\mathbf{h}_3 = (2, \text{big}, \text{no})$ be three hotels. $\mathbf{h}_1$ and $\mathbf{h}_2$ differ by just one value; they define a swap, denoted by $(\mathbf{h}_1, \mathbf{h}_2)_{\text{occupancy}}$. $\mathbf{h}_1$ and $\mathbf{h}_3$ (as well as $\mathbf{h}_2$ and $\mathbf{h}_3$) differ by more than one value, they do not define a swap. $\mathbf{h}_1[\text{occupancy}] = (2)$ and $\mathbf{h}_3[\text{occupancy}, \text{swimmingpool}] = (2, \text{no})$.

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 $V \in \mathbf{V}, Dom(V) = \{v, v'\}, \mathbf{U} = Pa(V) \subseteq \mathbf{V} \backslash \{V\}$, 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 $\mathbf{V}$: $CPT(\mathbf{V}) = \bigcup_{V \in \mathbf{V}} CPT(V)$.

Example 5 Considering Examples 2 and 3, we can formally define the preference rules for each variable, hence inducing a CP-table: suppose further that our couple will always choose a hotel with a swimming pool. Their whole preferences can now be formally represented by $CPT(\text{swimmingpool}) = \{(\text{yes} \succ \text{no})\}$, $CPT(\text{occupancy}) = \{(2 \succ 3)\}$. Finally, we know that $Pa(\text{kitchen}) = \{\text{occupancy}\}$ and $CPT(\text{kitchen}) = \{(2 : \text{small} \succ \text{big}), (3 : \text{big} \succ \text{small})\}$ that can also be represented by the following table:

$$\begin{array}{l} 2 : \text{small} \succ \text{big} \\ 3 : \text{big} \succ \text{small}. \end{array}$$

Definition 3 (Conditional preference network) 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 Fig. 2. One can note that it is a complete and a non-separable CP-net that contains an independent variable (swimming pool) and a variable conditioned by another one (kitchen is conditioned by occupancy).

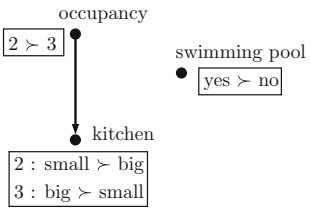

Fig. 2 Complete CP-net with three variables

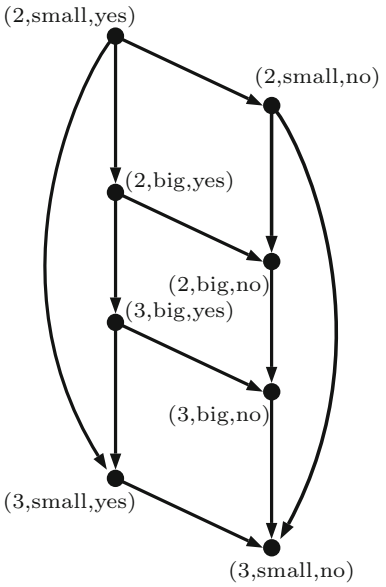

Fig. 3 Example of a partial ordering over the outcomes of the CP-net described in Fig. 2, from the best to the worst outcome

Each CP-net induces a partial ordering over its outcomes. One can easily deduce a preference between a swap $(\mathbf{o}, \mathbf{o}')_V$ by checking in the CP-net the validity of the rule on the variable V, i.e., $\mathbf{o} \succ \mathbf{o}'$ iff $\mathbf{o}[Pa(V)] : \mathbf{o}[\{V\}] \succ \mathbf{o}'[\{V\}]$. In such an ordering, two outcomes are adjacent iff they form a swap. If this condition holds, we connect these outcomes from the best to the worst preferred outcome. An example is given in Fig. 3 using Example 5 and its associated CP-net given in Fig. 2.

It is possible to obtain cycles in the outcomes preference ordering, which creates contradictions between preferences (e.g., $\mathbf{o}_1 \succ \mathbf{o}_2 \succ \mathbf{o}_1$). It is known from Boutilier et al. (2004) that for acyclic CP-nets, the associated outcomes' preference ordering is always acyclic. Moreover, given two different outcomes, finding the one that is preferred to the other one in cyclic CP-nets is PSPACE-hard (Boutilier et al. 2004). Henceforth, we focus in this paper on acyclic CP-nets.

We finally define a **class of CP-nets**, denoted by $\mathcal{C}_{(.)}$, where (.) corresponds to a structural property of the CP-nets. For instance, the class $\mathcal{C}_{acy}$ corresponds to the class of all acyclic CP-nets, and the class $\mathcal{C}_{tree}$ refers to the class of all tree-shaped CP-nets, i.e., all acyclic CP-nets $\mathcal{N}$, such that $|Pa(V)| \leq 1, \forall V \in \mathbf{V}$, where $|Pa(V)|$ denotes the cardinality of $Pa(V)$. Note that $\mathcal{C}_{tree} \subset \mathcal{C}_{acy}$.

2.2 Query learning algorithms for CP-nets

We discuss, in this section, the main query learning algorithms for CP-nets Koriche and Zanuttini (2010); Guerin et al. (2013). We denote by n the number of variables of a CP-net, i.e., $|\mathbf{V}| = n$, by $\mathcal{P}$ the target preference model,[3] 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 Koriche and Zanuttini (2010) proceeds by asking different questions to a user to learn her preferences. The working assumption is that the aggregation of preferences of users is approximately representable by an acyclic CP-net structure. Two types of queries can be distinguished. For $\mathcal{P}$ and $\mathcal{P}'$ two preference models:

- **equivalence queries** $\texttt{EQ}(\mathcal{P}, \mathcal{P}')$ that return `TRUE` if $\mathcal{P}$ is equivalent to $\mathcal{P}'$[4], otherwise they return `FALSE` plus a preference counter example representing a violated rule;
- **membership queries** $\texttt{MQ}(\mathcal{P}, r)$, which return `TRUE` if the rule r is satisfied in $\mathcal{P}$ (denoted by $\mathcal{P} \models r$), otherwise they return `FALSE`.

In this paper, we try to fit the learned CP-net $\mathcal{N}_L$ with the preference model $\mathcal{P}$ from an oracle Σ (see Definition 4), then when $\mathcal{N}_L \neq \mathcal{P}$, the equivalence query returns a counterexample swap $(\mathbf{o}, \mathbf{o}')_V$. This swap induces a rule r. Two cases are, therefore, possible:

(i) if $r \neq CPT(V)$, then we will simply add r to $\mathcal{N}_L$;
(ii) r is violated in $\mathcal{N}_L$, i.e., $\bar{r} \in CPT(V)$, then we need to find a new parent variable for V.

In Koriche and Zanuttini (2010), 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 Koriche and Zanuttini (2010).

The second algorithm (Guerin et al. 2013) is neither based on *ceteris paribus* comparisons nor equivalence and membership queries. It is an online algorithm that learns an acyclic CP-net and is decomposed into two phases:

1. finding a separable CP-net by asking for each variable V the preference between the assignments v and v',
2. updating the CP-table of each variable by finding the best set of parent variables using a set of confident[5] 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.,

[3] A preference model $\mathcal{P}$ corresponds to a model where the preferences have properties, such that conditional preferences, additive preferences, etc.

[4] We say that two preference models $\mathcal{P}$ and $\mathcal{P}'$ are equivalent, denoted by $\mathcal{P} \equiv \mathcal{P}'$ iff they induce exactly the same preferences.

[5] We consider a variable V as **confident** if enough swaps that induce the rules of V are found.

$\mathbf{P} \subseteq \mathbf{C} \subseteq \mathbf{V}$. 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. 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 preference model $\mathcal{P}$, 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 Guerin et al. (2013).

3 Proposed algorithm for learning a CP-net

Let $r = (\mathbf{u} : v \succ v')$ be a rule with $V \in \mathbf{V}, Dom(V) = \{v, v'\}, Pa(V) = \mathbf{U} \subseteq \mathbf{V} \backslash \{V\}$, 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 the rule r with an assignment $p \in Dom(P)$ of a new parent variable $P \in \mathbf{V} \backslash (\mathbf{U} \cup \{V\})$.

Definition 4 (Oracle) In this paper, an **oracle** Σ is a finite database which contains the preferences of a group of users (i.e., a set of swaps). This database can

- answer to a membership query by finding the given preference;
- answer to an equivalence query by comparing itself with the current learned CP-net, and by returning the first found counterexample swap if it exists.

The oracle is a central notion in query learning algorithms. We suppose here that each of the users who expresses her preferences can have its own preference model in her mind, which is not necessary a CP-net. Moreover, as the database (i.e., the oracle) is finite, it cannot necessarily contain all of the possible swaps.

As in Algorithm 1 in Koriche and Zanuttini (2010), we query an oracle Σ through $\texttt{EQ}(\mathcal{N}_L, \mathcal{P})$ if our learned CP-net $\mathcal{N}_L$ is equivalent to the induced (the target) preference model $\mathcal{P}$ of oracle Σ.

The target preference model $\mathcal{P}$ cannot generally be explicitly known due to, e.g., cognitive overloads for users or the size of the database. However, we suppose that the oracle is able to differentiate its proper preference model from the learned CP-net.

Our algorithm, in contrast with the state-of-the-art approaches, tries to take into account contradictory preferences due to the multiple users databases that we manipulate. These contradictions can be the result of

(i) different opinions between users;
(ii) and/or a user preference model which is not necessary compatible with a CP-net.

Hence, the learning procedure should be robust as much as possible to such contradictions. In our approach, we introduce a **list of violated rules** L that cannot be represented in our CP-net, because we cannot add a parent to a variable to represent the rule in $\mathcal{N}_L$. Then, we say that two CP-nets $\mathcal{N}_L$ and $\mathcal{P}$ 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 the 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 the exact learning theory (Angluin 1987), 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 $\mathcal{N}_L$.

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

Data: An oracle Σ which induces a target preference model $\mathcal{P}$.
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** $(\neg\texttt{EQ}(\mathcal{N}_L, \mathcal{P}, L))$ **do**
4 Let $(\mathbf{o}, \mathbf{o}')_V \in \Sigma$ be a counterexample swap 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** $\mathcal{N}_L$;

$\mathcal{N}_L$ is updated by the rule r induced by the counterexample swap provided by the oracle. As in Boutilier et al. (2004), two cases can occur: the inverse rule $\bar{r}$ is not present in $\mathcal{N}_L$, and 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 contradictions 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 $\bar{r}^{p'}$.

The algorithms in Guerin et al. (2013) and Koriche and Zanuttini (2010) try to compute the complete CP-table of V and restart for each new parent variable, which

leads to heavy computations. Nevertheless, it is realistic to assume that in real-world applications, CP-tables are not generally complete.

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. These local decisions have a consequence that the number of rules in the CP-table of the current variable can be multiplied by two in the worst case.

The procedure `searchParent` needs as an input the counterexample swap 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 the variable V. The variable P must satisfy the following conditions:

1. It should preserve the assignment p between two comparable[6] 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 $(\mathbf{o}'', \mathbf{o}''')_V$ be two swaps, with $V, P \in \mathbf{V}$, such that

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

These constraints are modeled in Line 1 of Algorithm 2. 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 counterexample swap, 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:

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

In case of equality, we pick one of them at random.

Example 6 Back to the hotel example, suppose that we want to deduce that the variable "swimming pool" is not a parent of the variable "size of the kitchen" (i.e., it cannot condition the size of the kitchen). Then, we need to receive one of these swaps:

– $\mathbf{o} = \{2, \text{small}, \text{yes}\} \succ \mathbf{o}' = \{2, \text{big}, \text{yes}\}$, and $\mathbf{o}'' = \{2, \text{small}, \text{no}\} \succ \mathbf{o}''' = \{2, \text{big}, \text{no}\}$. We can observe here that the assignments of swimming pool have no incidence on the preference over the assignments of kitchen;

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

– $\mathbf{o} = \{2, \text{small}, \text{yes}\} \succ \mathbf{o}' = \{2, \text{big}, \text{yes}\}$, and $\mathbf{o}'' = \{3, \text{big}, \text{yes}\} \succ \mathbf{o}''' = \{3, \text{small}, \text{yes}\}$. We can observe here that for the same assignment of swimming pool, the preference over the assignments of kitchen can vary.

Our parent search procedure is given in Algorithm 2. The parent detection is made by the existence of the swap $(\mathbf{o}'', \mathbf{o}''')_V$ and the application of Eq. (1) (we look for a parent variable that possesses such a swap). Furthermore, the contradiction detection is made by looking for the candidate parent variable (in the set $\mathbf{P}$) which minimizes the number of violated swaps (Eq. (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 $\mathbf{P} \leftarrow \{P \in \mathbf{V} \setminus (\{V\} \cup Pa(V)) \mid \neg\texttt{cycle}(\mathcal{N} = (\mathbf{V}, A \cup \{(P, V)\}, CPT(\mathbf{V}))$ and $\exists (\mathbf{o}'', \mathbf{o}''')_V \in \Sigma$ s.t. Eq. (1) returns `TRUE`$\}$;
2 **if** $(\mathbf{P} \neq \emptyset)$ **then**
3 **return** $\underset{P \in \mathbf{P}}{\text{argmin}}\{\#(\mathbf{o}'', \mathbf{o}''')_V \in \Sigma \mid \text{Eq. (2) returns } \texttt{TRUE}\}$;
4 **else return** "parent not found";

Example 7 Let us now try to learn a simple complete CP-net from Fig. 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 counterexample swap $((2, \text{big}), (3, \text{big}))_{occupancy}$. The algorithm finds the rule $r = (\emptyset : 2 \succ 3)$ that is added to $\mathcal{N}_L$. The next step is the equivalence query. Consider now that they answer the query by returning $((2, \text{small}), (2, \text{big}))_{kitchen}$. Then, the rule $r = (\emptyset : \text{small} \succ \text{big})$ 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 counterexample swap $((3, \text{big}), (3, \text{small}))_{kitchen}$ that induces the rule $r = (\emptyset : \text{big} \succ \text{small})$. Since the inverse rule $\bar{r} = (\emptyset : \text{small} \succ \text{big})$ already exists, the `searchParent` returns only the occupancy variable. These two new rules $r^3 = (3 : \text{big} \succ \text{small})$ and $\bar{r}^2 = (2 : \text{small} \succ \text{big})$ are then added in the CP-net.

We end this subsection by giving some complexity results, and commenting on the importance of the answers ordering. We suppose here that the target preference model is a CP-net denoted by $\mathcal{N}_T$.

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 in Algorithm 2 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 $O((n-1)(s+n^2)) \subset O(n^3+ns)$.

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 If $\mathcal{N}_T \in \mathcal{C}_{tree}$, then Algorithm 1 computes $\mathcal{N}_L$ with $2n$ equivalence queries and has a complexity of $O(n^3+n^2s+ne)$, where e is the time taken by `EQ` to return `True` or `False`.

Proof We need n equivalence queries to learn a separable CP-net, so it is in $O(ne)$. Furthermore, as $\mathcal{N}_T$ has a tree structure, there are at most $n-1$ parents (one for each variable without one leaf in the worst case) to find and there are at most $(n-1)$ arcs in $\mathcal{N}_L$, so `searchParent` has a complexity of $O(n^2+ns)$. Finally, we need $O(n^3+n^2s+ne)$ to compute $\mathcal{N}_L$, with $2n$ equivalence queries (n for learning the separable part of $\mathcal{N}_T$ and n for the adding parents part) □

Proposition 3 Algorithm 1 has a complexity of $O(2^p(n^4+n^2s+ne))$ 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 Proposition 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 that 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 $2n(2^p-1)(n^3+ns+e) \in O(2^p(n^4+n^2s+ne))$ □

Note that s and e, which are, respectively, defined in Propositions 1 and 3, correspond to the same process if the oracle is a database. For n the number of variables, the maximum number of objects in a database 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 database) and the worst case does not hold in most of situations.

An important remark about Algorithm 1 is that the order of the received counterexample swaps can perturb the choice of the parent variables during the learning phase.

Example 8 Consider a world $\mathbf{V} = \{A, B, C\}$ (with $Dom(A) = \{a, a'\}$, $Dom(B) = \{b, b'\}$ and $Dom(C) = \{c, c'\}$). Consider further that the oracle Σ will express its counterexample swaps in the following order: we receive $abc \succ abc'$ and deduce the rule $(\emptyset : c \succ c')$. Then, we receive $a'b'c' \succ a'b'c$, and we suppose that Algorithm 1

consider the variable B as a parent variable of C. We finally receive the counterexample swap $a'bc' \succ a'bc$ which means that the real parent variable of C is A.

The previous example shows us that, when a choice is possible between two variables, it is possible to choose a bad parent variable, which would not have been occur with the order $abc \succ abc', a'bc' \succ a'bc, a'b'c \succ a'b'c'$. To overcome this:

1. Algorithm 1 chooses the parent variable which **minimizes** the contradictions (Eq. (1)). This allows us to minimize the probability of choosing the bad parent;
2. considering the oracle Σ as a set of swaps (see Sect. 4), it is possible to generate k different databases by picking at random k' different swaps from Σ, to learn one CP-net per database, and select the CP-net which minimizes the number of contradictions between the learned CP-nets.

4 Experimental results

To evaluate the efficiency of our algorithm, two experimental protocols have been designed: the first one aimed at comparing our approach with the algorithm given in Guerin et al. (2013); the second one consists in demonstrating the usefulness of our algorithm on real and on simulated databases.

Defining an accuracy measure of a learning algorithm on such a structure is not a trivial task. There exists three different measures to define the accuracy of a learning algorithm for CP-nets:

- a similarity measure which computes the difference between the digraphs of the learned CP-net and the target one Liu et al. (2014). However, the target model, in our case, is not necessary a CP-net;
- an accuracy measure consisting in comparing the rules between the learned CP-net and the target one. However, even if this is relevant in some cases, where there is just one violated rule, this measure can be meaningless in other cases. Indeed, a violated rule cannot be represented in $\mathcal{N}_L$ and then induces an exponential number of unsatisfiable swaps, which is intractable;
- an accuracy measure consisting in comparing the number of swaps in the database Σ that are in agreement with $\mathcal{N}_L$ versus the total number of swaps in the whole database (Guerin et al. 2013; Liu et al. 2014; Michael and Papageorgiou 2013).

In our experiments, we consider the last measure to compute the accuracy $Acc(\mathcal{N}_L, \Sigma)$ of Algorithm 1 for the learned CP-net $\mathcal{N}_L$ with respect to a database (oracle) Σ:

$$Acc(\mathcal{N}_L, \Sigma) = \frac{|\{\mathbf{o} \succ_\Sigma \mathbf{o}' | \mathbf{o} \succ_{\mathcal{N}_L} \mathbf{o}'\}|}{|\{\mathbf{o} \succ_\Sigma \mathbf{o}'\}|},$$

where $\mathbf{o}$ and $\mathbf{o}'$ are a *ceteris paribus* pair, $|E|$ represents the cardinality of the set E, and $\succ_{\Sigma}$ (resp. $\succ_{\mathcal{N}_L}$) represents the preference relation in the database Σ (resp. in the outcomes preference ordering of $\mathcal{N}_L$).

Following Guerin et al. (2013), we consider the indecision case, denoted by $\mathbf{o} \bowtie \mathbf{o}'$, meaning that one cannot decide whether $\mathbf{o} \succ \mathbf{o}'$ or $\mathbf{o}' \succ \mathbf{o}$. These different possibilities are given in Table 1.

We use two different runs for our algorithm to smooth the results: the first one consists of the random generation phase of the target CP-nets or the databases, 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.

4.1 Learning a target CP-net $\mathcal{N}_T$

In this subsection, we suppose that the target preference model is a CP-net, which is denoted by $\mathcal{N}_T$. Then, our objective is to learn a CP-net $\mathcal{N}_L$ from a target CP-net $\mathcal{N}_T$. We suppose here that the oracle Σ is capable to express its CP-net $\mathcal{N}_T$. This target CP-net $\mathcal{N}_T$ is generated by the algorithm `GenerateRandomCPNet` given in Subsection 5.1 in Guerin et al. (2013). This algorithm generates an acyclic, binary, and complete random CP-net given a set of binary variables, the number of edges in the CP-net, and the maximum number of parent variables. We begin by creating all the conditional variables, and then, we generate a CP-table for every $V \in \mathbf{V}$: for each of the $2^{Pa(V)}$ possible values, we randomly add either $v \succ v'$ or $v' \succ v$.

We compare our algorithm with the one in Guerin et al. (2013). It is important to keep in mind that the algorithm in Guerin et al. (2013) is an **online** algorithm, while ours is an **offline** one. An online algorithm has to learn a structure with little information, whereas we actually explore all the available information. To make the comparison as fair as possible, we set the maximum number of parents to 5 and the density as $\delta = \frac{\#\text{arcs}}{n}$, which is the ratio of arcs per node of the target CP-net $\mathcal{N}_T$ ($\delta = \infty$ corresponds to the complete acyclic CP-net with respect to the parent condition). Furthermore, we suppose here to have a **non-contradictory** database Σ. Results of this experiment are summarized in Table 2 [values depicted in the white cells come from Table 1 in Guerin et al. (2013)].

As one could have expected, Table 2 shows better results for our algorithm in all the cases. However, our disagreement cases grow faster than the ones in Guerin et al. (2013). One can observe that the algorithm in Guerin et al. (2013) establishes

Table 1 Different possible comparisons between a target preference model $\mathcal{P}$ and a learned CP-net $\mathcal{N}_L$

$\mathcal{N}_L \backslash \mathcal{N}_T$	$\mathbf{o} \succ \mathbf{o}'$	$\mathbf{o}' \succ \mathbf{o}$	$\mathbf{o} \bowtie \mathbf{o}'$
$\mathbf{o} \succ \mathbf{o}'$	Agrees	Disagrees	Indecisive
$\mathbf{o}' \succ \mathbf{o}$	Disagrees	Agrees	Indecisive

The objective is to maximize the agreement

Table 2 Comparison results between our algorithm (gray cells) and algorithm of Guerin et al. (2013) (white cells)

n	Agreement	Disagreement	Indecision
$\delta = 1$			
4	0.98	0.02	0.00
	1.00	**0.00**	0.00
8	0.77	0.02	0.21
	0.98	0.02	**0.00**
12	0.65	**0.02**	0.34
	0.97	0.03	**0.00**
$\delta = 3$			
4	0.98	0.02	0.00
	1.00	**0.00**	0.00
8	0.80	**0.04**	0.16
	0.93	0.07	**0.00**
12	0.62	**0.04**	0.34
	0.90	0.10	**0.00**
$\delta = \infty$			
4	0.98	0.02	0.00
	0.99	**0.01**	0.00
8	0.76	**0.02**	0.22
	0.91	0.09	**0.00**
12	0.54	**0.04**	0.42
	0.86	0.14	**0.00**

Results are averaged on 10×3 runs and the best one, for each test, is written in bold

an equilibrium between agreement and indecision. This is due to the fact that when a new rule r has not enough information to be accepted, neither r nor $\bar{r}$ is added to $\mathcal{N}_L$. Our algorithm has no indecision, because the oracle always returns comparable outcomes.

Let us compare the algorithms in terms of speed. The results are summarized in Fig. 4.

The computation time of our algorithm exponentially grows with respect to the density δ and the number of variables n (see Fig. 4). We made these experiments on a laptop with an Intel core i7-4600U with 16Go of RAM (algorithms are implemented in Python). The experiment conducted in Guerin et al. (2013) considered fixed $\delta = 1$, $p = 12$, while n varies. The learning procedure takes less than 1 s for the worst test, while we need more than 2 s to do the same for $n = 12$. However, we need about 10 s to learn a CP-net that has 12 variables and a maximum density graph.

Let us now evaluate the performance of our algorithm when the number of parents in $\mathcal{N}_T$ and the number of variables is fixed (here, we, respectively, set 5 and 12), and the number of parents in $\mathcal{N}_L$ and the ratio vary. Figure 5 shows that the accuracy increases faster when the number of parents of $\mathcal{N}_L$ and $\mathcal{N}_T$ are the same. However, in case of separable CP-nets ($p = 0$), a sparse[7] CP-net ($\delta = 1$) can be

[7] We say that a CP-net is **sparse** if the adjacency matrix of its corresponding graph is sparse.

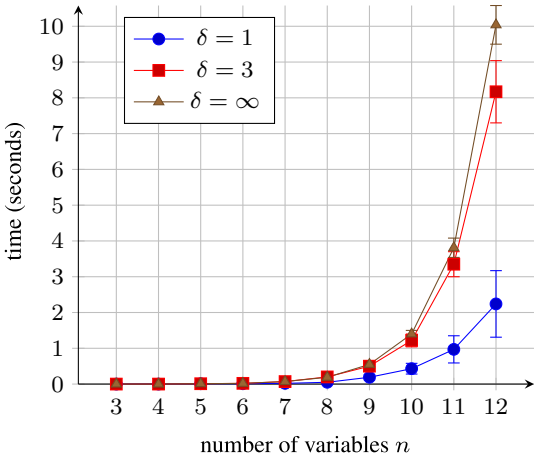

Fig. 4 Learning time according to the number of variables n. We set the number of parents to $p = 5$. Results correspond to one learning execution averaged on 10×3 runs and error bars correspond to the standard deviation of the observed values

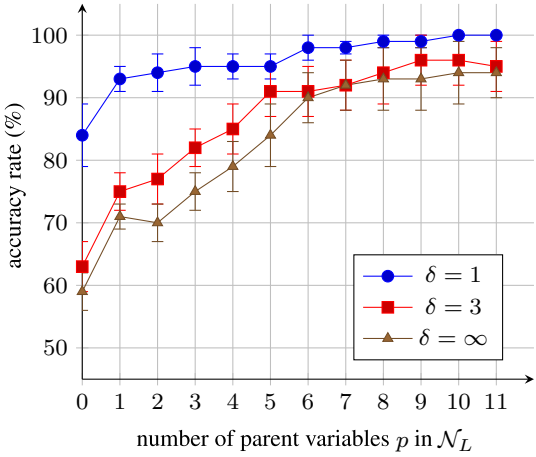

Fig. 5 Learning accuracy according to the number of parent variables p. We set the number variables to $n = 12$ and the maximum number of parents in $\mathcal{N}_T$ to 5. Results are averaged on 10×3 runs and error bars correspond to the standard deviation of the observed values

easily learned (more than 80% of accuracy) and their minimum and maximum values have a bounded variability, contrary to dense CP-nets ($\delta \geq 3$). Furthermore, a gap between the accuracy of separable and tree-shaped CP-nets is observed. This can be explained by the fact that the number of possible rules doubles between both (hence, more preferences can be represented). However, the accuracy decreases when p changes from 1 to 2. This new parent variable may increase the complexity of the CP-net structure. Thus, a rule inferred when $p = 1$ could be modified when $p = 2$.

4.2 Learning a CP-net from a database

In this section, we consider that the oracle Σ is a database that contains a list of swaps.

We use two distinct databases to conduct our experiments:

1. the TripAdvisor database (Wang et al. 2010, 2011) rescaled to obtain binary attributes (considered here as variables),
2. randomly generated database to test the scalability of our algorithm, and its robustness to contradictions.

The TripAdisor[8] database contains about 240, 000 hotel reviews from about 1, 000 users. 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 preference model $\mathcal{P}$ that potentially contains contradictions.

We also generate a random artificial database 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. To test Algorithm 1, we generate three sizes of random databases with 50 objects (7 attributes), 500 objects (10 attributes), and 10,000 objects (15 attributes). We set the size of vectors of each database by applying $n = \lfloor \log_2 m \rfloor + 1$, with m the number of objects in the database. We suppose that such a generated synthetic database reflects reality in the sense that a real database 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.

We first test the importance of having coherent preferences for the learning phase. Except for the 50 objects database (probably due to the few number of objects), a gap is observed between the real hotel database and the 500 objects' database in Fig. 6. Of course, the agreement grows with regards to the maximum number of parents. The last test consists in relaxing the acyclic condition to observe the its influence. In this case, we are able to exactly fit $\mathcal{P}$, but at the price of heavy computations.

Figure 6 depicts the results of Table 3. The accuracy increases linearly with respect to 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 databases, 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 for small numbers of objects ($n < 500$) because of the number of variables that does not allow the algorithm to correct the contradictions.

Figure 7 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 databases). When the database 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 Fig. 6, with a negative gap between $p = 1$ and $p = 2$. It occurs for all random databases.

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

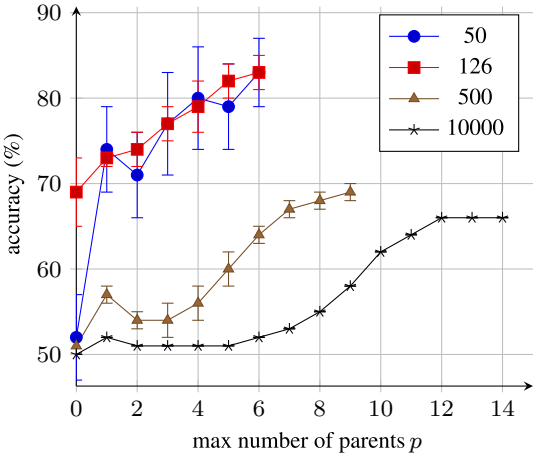

Fig. 6 Learning accuracy according to the number of parent p per variable. Databases are randomly generated except for the real 126 hotels file. Results are averaged on 5×10 runs and error bars correspond to the standard deviation of the observed values

Table 3 Accuracy of Algorithm 1

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 (10,000)	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 (10,000)	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 (10,000)	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 (10,000)	*Too long to compute*	

Results are averaged on 10×10 runs

Another remark concerns the impact of the order of the answers of the oracle when Algorithm 1 receives the counterexample swaps. We observe that, for a large number of swaps, the standard deviation is low (accuracy value ± 2). This experimentally demonstrates that, in practice, for a large number of swaps, their orders have a few impact on the quality of the learning procedure.

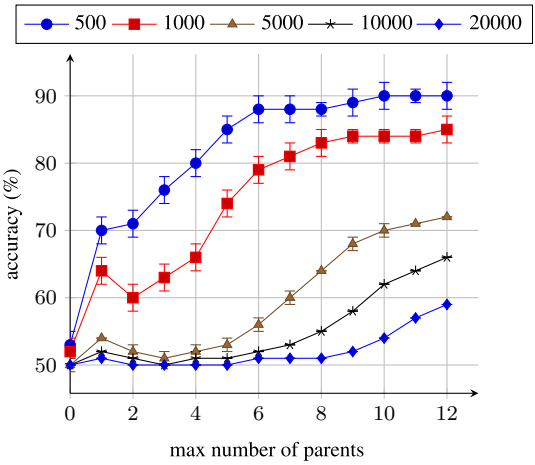

Fig. 7 Learning accuracy when the number of variables is fixed ($n = 15$). We increase the number of objects in the random databases. Results are averaged on 2×5 runs and error bars correspond to the standard deviation of the observed values

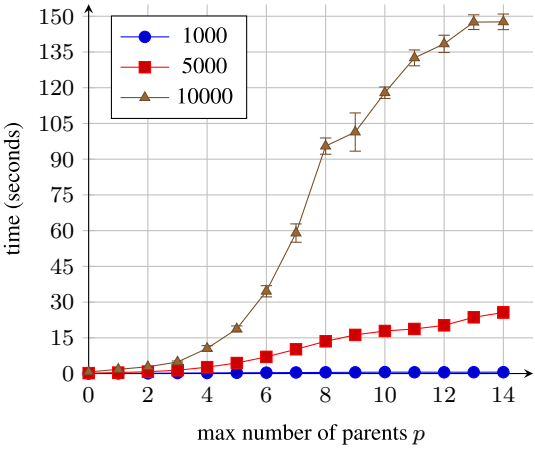

Fig. 8 Learning time when the number of variables is fixed ($n = 15$). We increase the number of objects in the databases. Results correspond to one learning execution averaged on 2×5 runs and error bars correspond to the standard deviation of the observed values

At last, we focus on the time taken by Algorithm 1 to learn one CP-net from a database. We can see in Fig. 8 that this learning task is immediate for $m \leq 1000$. However, whereas we need about 20 s to learn a CP-net from $m = 5000$ and $p = 14$, we need more than 150 s to learn a CP-net from $m = 10{,}000$ and $p = 14$. Thus, for two times more objects, we need ten times longer to compute 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 databases, and with respect to the state of the art. They showed that despite its

exponential complexity depending on the number of objects and parents (Proposition 3), our algorithm can learn in a few seconds a CP-net on random CP-nets, random databases, or real database. Besides, when compared to the online algorithm in Guerin et al. (2013), our algorithm gives better results in terms of accuracy, but it generally needs more time to return a CP-net. This result is expected, since our algorithm is an offline method, while the one in Guerin et al. (2013) follows an online approach.

The main ingredient of our approach is contradiction handling in the preferences. The robustness of our algorithm to these contradictions has been evidenced by the designed experiments. For instance, on a task for hotel rating using a TripAdvisor database that contains multiple users, and then, many contradictions, 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 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 databases 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 $\mathcal{P}$ and $\mathcal{N}_L$ (when working with databases), 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 rule generation.

References

Alanazi E, Mouhoub M, Zilles S (2016) 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. http://www.ijcai.org/Abstract/16/196

Angluin D (1987) Queries and concept learning. Mach Learn 2(4):319–342. doi:10.1007/BF00116828

Boutilier C, Brafman RI, Domshlak C, Hoos HH, Poole D (2004) Cp-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. J Artif Intell Res 21:135–191. doi:10.1613/jair.1234

Chevaleyre Y, Koriche F, Lang J, Mengin J, Zanuttini B (2010) Learning ordinal preferences on multiattribute domains: the case of cp-nets. In: Preference Learning, pp 273–296. doi:10.1007/978-3-642-14125-6_13

Dimopoulos Y, Michael L, Athienitou F (2009) Ceteris paribus preference elicitation with predictive guarantees. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, California, USA, 11–17 July 2009, pp 1890–1895. http://ijcai.org/Proceedings/09/Papers/313.pdf

Eckhardt A, Vojtás P (2009) How to learn fuzzy user preferences with variable objectives. In: Proceedings of the Joint 2009 International Fuzzy Systems Association World Congress and 2009

European Society of Fuzzy Logic and Technology Conference, Lisbon, Portugal, 20–24 July 2009, pp 938–943. http://www.eusflat.org/proceedings/IFSA-EUSFLAT_2009/pdf/tema_0938.pdf

Eckhardt A, Vojtás P (2010) Learning user preferences for 2cp-regression for a recommender system. In: SOFSEM 2010: Theory and Practice of Computer Science, 36th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, 23–29 January 2010, pp 346–357. doi:10.1007/978-3-642-11266-9_29

Fürnkranz J, Hüllermeier E (eds) (2010) Preference Learning. Springer, Berlin. doi:10.1007/978-3-642-14125-6

Guerin JT, Allen TE, Goldsmith J (2013) Learning cp-net preferences online from user queries. In: Late-Breaking Developments in the Field of Artificial Intelligence, Bellevue,Washington, USA, 14–18 July 2013. http://www.aaai.org/ocs/index.php/WS/AAAIW13/paper/view/7114

Koriche F, Zanuttini B (2010) Learning conditional preference networks. Artif Intell 174(11):685–703. doi:10.1016/j.artint.2010.04.019

Liu J, Yao Z, Xiong Y, LiuW,Wu C (2013) Learning conditional preference network from noisy samples using hypothesis testing. Knowl Based Syst 40:7–16. doi:10.1016/j.knosys.2012.11.006

Liu J, Xiong Y, Wu C, Yao Z, Liu W (2014) Learning conditional preference networks from inconsistent examples. IEEE Trans Knowl Data Eng 26(2):376–390. doi:10.1109/TKDE.2012.231

Liu J, Sui C, Deng D, Wang J, Feng B, Liu W, Wu C (2016) Representing conditional preference by boosted regression trees for recommendation. Inf Sci 327:1–20. doi:10.1016/j.ins.2015.08.001

Michael L, Papageorgiou E (2013) An empirical investigation of ceteris paribus learnability. In: IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, 3–9 August 2013, pp 1537–1543. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6923

Tsoukiàs A (2008) From decision theory to decision aiding methodology. Eur J Operat Res 187(1):138–161. doi:10.1016/j.ejor.2007.02.039

Wang H, Lu Y, Zhai C (2010) 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, Washington, DC, USA, 25–28 July 2010, pp 783–792. doi:10.1145/1835804.1835903

Wang H, Lu Y, Zhai C (2011) Latent aspect rating analysis without aspect keyword supervision. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, 21–24 August 2011, pp 618–626. doi:10.1145/2020408.2020505