Ceteris Paribus Preference Elicitation with Predictive Guarantees

Yannis Dimopoulos and **Loizos Michael** and **Fani Athienitou**
Department of Computer Science, University of Cyprus
CY-1678 Nicosia, Cyprus
{yannis, loizosm}@cs.ucy.ac.cy and fani.athienitou@gmail.com

Abstract

CP-networks have been proposed as a simple and intuitive graphical tool for representing conditional ceteris paribus preference statements over the values of a set of variables. While the problem of reasoning with CP-networks has been receiving some attention, there are very few works that address the problem of learning CP-networks.

In this work we investigate the task of learning CP-networks, given access to a set of pairwise comparisons. We first prove that the learning problem is intractable, even under several simplifying assumptions. We then present an algorithm that, under certain assumptions about the observed pairwise comparisons, identifies a CP-network that entails these comparisons. We finally show that the proposed algorithm is a PAC-learner, and, thus, that the CP-networks it induces accurately predict the user's preferences on previously unseen situations.

1 Introduction

Preferences have been studied for a long time in different scientific fields such as economics, operations research, decision theory, and computer science. One of the central problems there is that of automated learning and adaptation of preferences. The significance of this problem has been recognized quite early, and there is now a considerable body of research related to the problem of *preference elicitation*. Within Artificial Intelligence, some aspects of the problem of preference elicitation have more recently received some attention in Machine Learning. Among the different problems that have been investigated, that of *label ranking* (see, e.g., [Brinker *et al.*, 2006; Cohen *et al.*, 1999]) has some similarities with the problem we study in this work. We direct the interested reader to the relevant literature for more details.

CP-networks (CP-nets for short) have been proposed as a simple and intuitive graphical tool for representing and reasoning with qualitative conditional preferences. CP-nets are directed graphs whose vertices correspond to variables, each annotated with preferences over the variable's values. These preferences may depend on the values of other variables, hence the term conditional. Although CP-nets, being a compact, intuitive language for representing preferences, alleviate the problem of preference elicitation, they do not completely eliminate it. Indeed, deriving a CP-net in complex domains can be a tedious and time-consuming task. Moreover, in some domains, such as auctions and automated negotiation, users may not be willing to reveal their preferences. Thus, there is a need for deriving methods for learning CP-nets.

Despite the evident significance of the problem, there are only a couple of studies that address the question of learning CP-networks, namely [Athienitou and Dimopoulos, 2007] and [Lang and Mengin, 2008]. In this work we extend and complement these previous studies in several ways. Specifically, we study the problem of deriving a CP-net from a given set $\mathcal{P}$ of pairwise comparisons $o_i \succ o_j$ between outcomes, meaning "outcome o_i is *strictly* preferred over outcome o_j". Such pairwise comparisons may be gathered, for instance, by passively observing the choices of a user on a web page. In the simplest case, we seek to derive a CP-net N such that $\mathcal{P}$ is a subset of the relation induced by N. We show that the problem is intractable even in the case where the input comparisons are on outcomes that differ on at most two variables, and the in-degree of the nodes of the simplest CP-net that implies the comparisons (if one exists) is upper-bounded by 2.

Although CP-net learning is hard in the general case, we present an algorithm that solves this problem in polynomial time, provided that some restrictions on the input pairwise comparison and the CP-net to be learned are satisfied. Moreover, we show the the proposed algorithm is a PAC-learner, i.e., the CP-networks it induces accurately predict the user's preferences on future and possibly previously unseen variable assignments. A reduction suggests the hardness of improving our algorithm's running time, and establishes the fixed-parameter intractability of the learning problem.

2 CP-networks

We start by briefly reviewing the basics of CP-nets, following [Boutilier *et al.*, 2003]. Let $\mathcal{V} = \{X_1, \cdots, X_n\}$ be a set of variables over which a user has preferences. Each variable X_i is associated with a domain of values $\mathrm{Dom}(X_i) = \{x_1^i, \cdots, x_{n_i}^i\}$. An assignment x of values to a set $\mathcal{X} \subseteq \mathcal{V}$ of variables maps each variable of $\mathcal{X}$ to an element of its domain; if $\mathcal{X} = \mathcal{V}$, x is an *outcome*. Denote $\mathrm{Asst}(\mathcal{X})$ the set of all assignments to $\mathcal{X} \subseteq \mathcal{V}$. For $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{V}$ such that $\mathcal{X} \cap \mathcal{Y} = \emptyset$, $xy \in \mathrm{Asst}(\mathcal{X} \cup \mathcal{Y})$ denotes the combination of assignments $x \in \mathrm{Asst}(\mathcal{X})$ and $y \in \mathrm{Asst}(\mathcal{Y})$.

A *preference ranking* is a total preorder $\succeq$ over the set of outcomes. The notation $o_1 \succeq o_2$ means that outcome o_1 is equally or more preferred than o_2, while $o_1 \succ o_2$ means that outcome o_1 is strictly more preferred by the decision maker than o_2 (i.e., $o_1 \succeq o_2$ and $o_2 \not\succeq o_1$). Finally, $o_1 \sim o_2$ denotes that outcomes o_1 and o_2 are indifferent.

A CP-net N is constructed as follows. For each variable X_i, the user specifies a set of parent variables $\mathrm{Pa}_N(X_i)$ that affect her preference over the values of X_i. Then, she specifies her preferences over the values of X_i for all instantiations of the variable set $\mathrm{Pa}_N(X_i)$. More specifically, for each assignment $u \in \mathrm{Asst}(\mathrm{Pa}_N(X_i))$, the user provides in a *conditional preference table* or CPT, a total preorder $\succeq^i_u$ over $\mathrm{Dom}(X_i)$ by specifying for any two values x and x', either $x \succ^i_u x'$, $x' \succ^i_u x$, or $x \sim^i_u x'$. For simplicity of presentation, we ignore indifference for the rest of this paper.

Definition 1. *A **CP-net** N over the set $\mathcal{V} = \{X_1, \cdots, X_n\}$ of variables is a directed graph G over the vertex set $\mathcal{V}$, with each $X_i \in \mathcal{V}$ annotated with a conditional preference table* $\mathrm{CPT}(X_i)$. *Each conditional preference table* $\mathrm{CPT}(X_i)$ *associates a total order $\succ^i_u$ with each $u \in \mathrm{Asst}(\mathrm{Pa}_N(X_i))$.*

The following notions define the semantics of a CP-net.

Definition 2 ([Boutilier *et al.*, 2003])**.** *Consider a CP-net N over the set $\mathcal{V}$ of variables, a variable $X_i \in \mathcal{V}$, and $\mathcal{Y} = \mathcal{V} \setminus (\mathrm{Pa}_N(X_i) \cup \{X_i\})$. For every $u \in \mathrm{Asst}(\mathrm{Pa}_N(X_i))$, let $\succ^i_u$ be the ordering over $\mathrm{Dom}(X_i)$ dictated by $\mathrm{CPT}(X_i)$. A preference ranking $\succ$ over $\mathrm{Asst}(\mathcal{V})$ **satisfies** (i) the total preorder $\succ^i_u$ iff $uxy \succ ux'y$ holds for every $y \in \mathrm{Asst}(\mathcal{Y})$, and every $x, x' \in \mathrm{Dom}(X_i)$ s.t. $x \succ^i_u x'$; (ii) the conditional preference table $\mathrm{CPT}(X_i)$ iff it satisfies $\succ^i_u$ for every $u \in \mathrm{Asst}(\mathrm{Pa}_N(X_i))$; (iii) the CP-net N iff it satisfies $\mathrm{CPT}(X_i)$ for every $X_i \in \mathcal{V}$. A CP-net N is **satisfiable** iff there exists a preference ranking $\succ$ that satisfies N.*

Definition 3. *Let N be a CP-net over a set $\mathcal{V}$ of variables, and let $o, o' \in \mathrm{Asst}(\mathcal{V})$ be any two outcomes. N **entails** $o \succ o'$, denoted $N \models o \succ o'$, iff $o \succ o'$ holds in every preference ranking that satisfies N.*

For the remainder of this paper, we restrict our attention to the class of acyclic CP-nets. We denote by $\mathcal{N}_k$ the subclass of acyclic CP-nets with a maximum in-degree at most k.

3 Learning by Fitting the Data

One of the simplest definitions of CP-net learnability, and the one considered here, is that of identifying a CP-net consistent with a given set of pairwise comparisons between outcomes that are entailed by some unknown target CP-net.

Definition 4 (CP-Net Consistent-Learnability)**.** *A class $\mathcal{N}$ of CP-nets over a set $\mathcal{V}$ of variables is **consistent-learnable (by** a class $\mathcal{H}$ of hypotheses) if there exists an algorithm $\mathcal{L}$ such that for every CP-net $N \in \mathcal{N}$, and every set $\mathcal{P}$ of pairwise comparisons between outcomes over $\mathcal{V}$ entailed by N, algorithm $\mathcal{L}$ has the following property: given access to $\mathcal{V}$ and $\mathcal{P}$, algorithm $\mathcal{L}$ runs in time polynomial in $|\mathcal{V}|$, $|\mathcal{P}|$, and $size(N)$, and returns a hypothesis h (in $\mathcal{H}$) that entails every comparison in $\mathcal{P}$.*

The constraint of Definition 4 that only polynomial time is allowed, makes learnability impossible in the general case.

Theorem 1. *Fix a constant integer $k \geq 2$. Deciding whether there exists a CP-net in $\mathcal{N}_k$ that entails all pairwise comparisons in a set $\mathcal{P}$ is* NP*-hard, even if for every $o_1 \succ o_2 \in \mathcal{P}$, o_1, o_2 differ on at most two variables, and even if when some CP-net entails all comparisons in $\mathcal{P}$, then there also exists a CP-net $N \in \mathcal{N}_2$ that entails all comparisons in $\mathcal{P}$.*

Proof (sketch). By reduction from read-3 3-SAT. Given a formula $\varphi = \bigwedge_{j=1}^{m} \ell_{j,1} \vee \ell_{j,2} \vee \ell_{j,3}$ over variables $\{v_1, \ldots, v_n\}$, define the set $\mathcal{V}$ to contain one variable for each variable v_i and clause c_j of φ, and their negations $\overline{v_i}$, $\overline{c_j}$. Without loss of generality, assume that every variable of φ that appears in some clause appears negated in some other clause (thus each literal appears at most twice), and not in the same clause.

Let the domain of $x \in \mathcal{V}$ be $\{x^0, x^1\}$, and a 0 in outcomes mean that all unspecified variables are false. For each c_j, $\mathcal{P}[\varphi]$ includes $c_j^1 \overline{c_j}^0 0 \succ c_j^0 \overline{c_j}^1 0$; roughly, this preference corresponds to asking that the clause c_j of φ should be satisfied.

We associate a set $\mathrm{PP}(x)$ of possible parents with each $x \in \mathcal{V}$. For each clause c_j, $\mathrm{PP}(c_j) = \{\ell_{j,1}, \ell_{j,2}, \ell_{j,3}\}$, $\mathrm{PP}(\overline{c_j}) = \{\}$. For each literal ℓ, $\mathrm{PP}(\ell) = \{\overline{\ell}\} \cup \{\overline{c_r} \mid c_r \text{ contains } \overline{\ell}\}$. We constrain CP-nets $N \in \mathcal{N}_k$ so that $\mathrm{Pa}_N(x) \cap \mathrm{PP}(x) \neq \emptyset$. We do so by excluding the possibility that any set $S \subseteq \mathcal{V}$ of k variables such that $\mathcal{S} \cap \mathrm{PP}(x) = \emptyset$ is the parent set $\mathrm{Pa}_N(x)$ of x. This is achieved if $\mathcal{P}[\varphi]$ includes $x^0 s 0 \succ x^1 s 0$ for every assignment $s \in \{0,1\}^k$ of values to $\mathcal{S}$. The construction can be done in time polynomial in the size of φ since k is constant.

It can be shown that the preferences in $\mathcal{P}[\varphi]$ are entailed by some CP-net N in $\mathcal{N}_k$ if and only if for each j, there exists t such that $\ell_{j,t}$ is a parent of c_j and a child of $\overline{\ell_{j,t}}$ in N. For the "only if" direction, the CPT of $x \in \mathcal{V}$ includes $x^0 \succ x^1$ when all its parents are false, and $x^1 \succ x^0$ otherwise. Thus, the parent relations in N induce satisfying assignments for φ, and vice versa, which suffices to establish the reduction. □

The following result is an immediate conclusion:

Corollary 1. *Fix a constant integer $k \geq 2$. The class $\mathcal{N}_2$ of CP-nets is not consistent-learnable by $\mathcal{N}_k$, unless* P = NP.

Thus, consistent-learnability is intractable given the standard complexity assumption P $\neq$ NP, even if the target CP-net is in $\mathcal{N}_2$, with binary variables, even if more expressive CP-nets may be returned (i.e., in $\mathcal{N}_k$), and even if outcomes in pairwise comparisons differ on at most two variables.

We present next an algorithm that tackles the problem of learning, as in Definition 4, from restricted sets $\mathcal{P}$. (The algorithm assumes binary variables, but the same basic idea, applies for non-binary variables.) The algorithm takes as input a set $\mathcal{V}$ of variables, and a set $\mathcal{P}$ of pairwise comparisons between outcomes over $\mathcal{V}$. It generates an *acyclic* CP-net that satisfies the given input, by creating the CPT of each variable.

The main procedure $learn(\mathcal{V}, \mathcal{P})$ is shown in pseudocode in Algorithm 1. The algorithm maintains a CP-net h that is initially empty, and attempts to extend it with the addition of the CPT of one of the remaining variables $\mathcal{X}$, by calling the procedure $extendNetwork(\mathcal{X}, \mathcal{V} \setminus \mathcal{X}, \mathcal{P}, k, h)$. If a CPT for each variable in $\mathcal{V}$ can be added, the constructed CP-net h is returned. Otherwise, the value of k is increased and the process loops, until either a CP-net is constructed,

or failure is detected. For each invocation of the procedure $extendNetwork(\mathcal{X}, \mathcal{V} \setminus \mathcal{X}, \mathcal{P}, k, h)$, all variables in $\mathcal{X}$ are tested until one is identified whose CPT can be constructed (if possible). For each candidate variable $X \in \mathcal{X}$, the procedure $getNextSubset(\mathcal{V}', k, \mathcal{U})$ is invoked to return the possible subsets $\mathcal{U}$ of $\mathcal{V}'$ of size at most k. Each such subset $\mathcal{U} \subseteq \mathcal{V}'$ is provisionally assumed to be the parent set of X. The procedure $createCPT(X, \mathcal{U}, \mathcal{P}, h)$ is then invoked, which checks the set of pairwise comparisons $\mathcal{P}$ to determine if $\mathrm{Pa}_h(X)$ can be set to $\mathcal{U}$ in the CP-net h being constructed. If so, the constructed $\mathrm{CPT}(X)$ is added in h, and the procedure returns the variable X. Otherwise, the next provisional parent set is considered. If $\mathrm{Pa}_h(X)$ cannot be set to any of the provisional parent sets $\mathcal{U}$, the next candidate variable is considered. If the CPT of no variable X can be added to the CP-net h, the empty set is returned.

Algorithm 1 $learn$ (variable set $\mathcal{V}$, comparison set $\mathcal{P}$)

```
h := the empty CP-net                    /* Constructed CP-net */
k := 0                                   /* Number of allowed parents */
X := V                                   /* Remaining variables not in h */
repeat
    repeat
        added := extendNetwork(X, V \ X, P, k, h)
        X := X \ added
    until added = ∅
    if X = ∅ then
        return h
    end if
    k := k + 1
until k = |V|
return failure
```

Algorithm 2 $extendNetwork$ (variable set $\mathcal{X}$, variable set $\mathcal{V}'$, comparison set $\mathcal{P}$, number k, CP-net h)

```
while X ≠ ∅ do
    choose X from X
    U := ∅                               /* Provisional parent set for X */
    repeat
        CPTcreated := createCPT(X, U, P, h)
        subsetExists := getNextSubset(V', k, U)
    until CPTcreated or not(subsetExists)
    if CPTcreated then
        return {X}
    end if
    X := X \ {X}
end while
return ∅
```

Observe that since $\mathcal{V}' = \mathcal{V} \setminus \mathcal{X}$, then $\mathcal{V}'$ contains exactly those variables that are already in the CP-net h at the time that the CPT of some variable $X \in \mathcal{X}$ is added. This guarantees that the constructed CP-net h is acyclic. Note further that we may assume that the subsets $\mathcal{U} \subseteq \mathcal{V}'$ returned by the procedure $getNextSubset(\mathcal{V}', k, \mathcal{U})$, are considered in order of increasing size. Although this is not required for our formal results to go through, it might provide a heuristic for further reducing the size of the constructed CP-nets.

To decide whether $\mathcal{U}$ can be the parent set of X in h, the procedure $createCPT(X, \mathcal{U}, \mathcal{P}, h)$ examines each $u_i x_i y_i \succ u_j x_j y_j \in \mathcal{P}$, where $u_i, u_j \in \mathrm{Asst}(\mathcal{U})$, $x_i, x_j \in \mathrm{Dom}(X)$, $y_i, y_j \in \mathrm{Asst}(\mathcal{V} \setminus (\mathcal{U} \cup \{X\}))$, and considers the inclusion of either $u_i : x_i \succ x_j$ or $u_j : x_i \succ x_j$ in $\mathrm{CPT}(X)$. Given the set

$$S = \{(u_i : x_i \succ x_j \textbf{ or } u_j : x_i \succ x_j) \mid u_i x_i y_i \succ u_j x_j y_j \in \mathcal{P}\}$$

of all choices that procedure $createCPT(X, \mathcal{U}, \mathcal{P}, h)$ has to make, the procedure attempts to generate $\mathrm{CPT}(X)$ by selecting one of the disjuncts in each element of S such that the selected entries are jointly consistent (i.e., for every choice of u, exactly one of $u : x_1 \succ x_2$ and $u : x_2 \succ x_1$ is chosen).

The above problem of selecting disjuncts from the set S can be naturally casted as a propositional satisfiability problem: to each element $u \in \mathrm{Asst}(\mathcal{U})$ that appears in S associate a boolean variable $v(u)$, whereas to each entry $u : x_1 \succ x_2$ associate the positive literal $v(u)$, and to each entry $u : x_2 \succ x_1$ the negative literal $\overline{v(u)}$. Then, the elements of S translate to binary clauses, and the resulting satisfiability problem is a 2-SAT one. The following example illustrates the translation.

Example 1. *Consider the variables A, B, C, with domains $\{a_1, a_2\}$, $\{b_1, b_2\}$, $\{c_1, c_2\}$, respectively. Consider also the set $\mathcal{P}$ of comparisons:*

(1) $a_1b_1c_1 \succ a_1b_2c_2$ *(2)* $a_1b_1c_2 \succ a_2b_1c_1$
(3) $a_2b_1c_1 \succ a_2b_1c_2$ *(4)* $a_2b_2c_1 \succ a_1b_2c_2$

During the invocation of $createCPT(C, \{A, B\}, \mathcal{P}, h)$, the set S of choices is populated as follows:

(1) $a_1b_1 : c_1 \succ c_2$ ***or*** $a_1b_2 : c_1 \succ c_2$
(2) $a_1b_1 : c_2 \succ c_1$ ***or*** $a_2b_1 : c_2 \succ c_1$
(3) $a_2b_1 : c_1 \succ c_2$ *(the second disjunct is the same)*
(4) $a_2b_2 : c_1 \succ c_2$ ***or*** $a_1b_2 : c_1 \succ c_2$

Assume that we associate the elements of $\mathrm{Asst}(\{A, B\})$ with boolean variables as follows: $a_1b_1 \to v_1$, $a_1b_2 \to v_2$, $a_2b_1 \to v_3$, $a_2b_2 \to v_4$. Then, the candidate entries are associated with literals as follows: $a_1b_1 : c_1 \succ c_2 \to v_1$, $a_1b_1 : c_2 \succ c_1 \to \overline{v_1}$, $a_1b_2 : c_1 \succ c_2 \to v_2$, $a_1b_2 : c_2 \succ c_1 \to \overline{v_2}$, $a_2b_1 : c_1 \succ c_2 \to v_3$, $a_2b_1 : c_2 \succ c_1 \to \overline{v_3}$, $a_2b_2 : c_1 \succ c_2 \to v_4$, $a_2b_2 : c_2 \succ c_1 \to \overline{v_4}$. The resulting 2-SAT instance is $(v_1 \vee v_2) \wedge (\overline{v_1} \vee \overline{v_3}) \wedge v_3 \wedge (v_4 \vee v_2)$.

A satisfying assignment for the above 2-SAT instance assigns $false$ to v_1, and $true$ to v_2, v_3 (and either truth-value to v_4). This translates to the CPT entries $a_1b_1 : c_2 \succ c_1$, $a_1b_2 : c_1 \succ c_2$, and $a_2b_1 : c_1 \succ c_2$ (and the remaining entry can be either $a_2b_2 : c_1 \succ c_2$ or $a_2b_2 : c_2 \succ c_1$).

Thus, constructing a CPT for a variable X given a provisional parent set $\mathcal{U}$ reduces to computing a satisfying assignment of a 2-SAT instance, which can be done in polynomial time [Aspvall *et al.*, 1979]. If the 2-SAT instance is unsatisfiable, $\mathcal{U}$ cannot be the parent set of X in the CP-net h.

We now illustrate the working of the entire algorithm.

Example 2. *Consider the set $\mathcal{V} = \{A, B, C, D\}$ of variables, with $\mathrm{Dom}(A) = \{a_1, a_2\}$, $\mathrm{Dom}(B) = \{b_1, b_2\}$, $\mathrm{Dom}(C) =$*

$\{c_1, c_2\}$, $\mathrm{Dom}(D) = \{d_1, d_2\}$. *Consider also the following set $\mathcal{P}$ of comparisons:*

(1) $a_1b_1c_2d_1 \succ a_1b_2c_2d_2$ *(2)* $a_1b_2c_1d_2 \succ a_1b_1c_1d_1$

(3) $a_2b_2c_2d_2 \succ a_2b_1c_2d_1$ *(4)* $a_1b_1c_1d_1 \succ a_1b_2c_1d_1$

(5) $a_1b_2c_1d_1 \succ a_2b_1c_2d_1$

During the algorithm's first iteration with $k = 0$, the algorithm successfully constructs CPTs for the variables A and C through two consecutive invocations of the procedure $extendNetwork$. The CP-net h is, thus, updated to include $: a_1 \succ a_2$ as $\mathrm{CPT}(A)$, and $: c_1 \succ c_2$ as $\mathrm{CPT}(C)$. During the third invocation of the procedure $extendNetwork$, the empty set is returned, since the CPT of none of the remaining variables B and D can be constructed with an empty parent set — in comparisons (1) and (2) the preference over b_1 and b_2 is reversed; so is the case for d_1 and d_2.

The value of k is increased to 1, and the algorithm attempts again to create CPTs for B and D, only to discover that this is still not possible. The provisional parent set $\{A\}$ does not become an actual parent set of B, since in comparisons (1) and (2) conditioning on a_1 does not resolve the fact that the preference over b_1 and b_2 is reversed. Similarly, the provisional parent set $\{C\}$ fails due to the comparisons (1) and (3), where conditioning on c_2 does not resolve the reversed preference over b_1 and b_2. An analogous situation applies when the algorithm attempts to build the CPT of D.

In the subsequent iteration the value of k is increased to 2. Assume that the algorithm first chooses the variable B. All attempts to create $\mathrm{CPT}(B)$ with a parent set other than $\{A, C\}$ fail as before. The attempt also fails when $\{A, C\}$ is considered, since in comparisons (2) and (4) conditioning on a_1c_1 does not resolve the reversed preference over b_1 and b_2. Next, variable D is chosen. When the procedure $createCPT(D, \{A, C\}, \mathcal{P}, h)$ is invoked, it succeeds and adds the following CPT as $\mathrm{CPT}(D)$ in the CP-net h:

a_1c_1	$d_2 \succ d_1$
a_1c_2	$d_1 \succ d_2$
a_2c_1	$d_1 \succ d_2$
a_2c_2	$d_2 \succ d_1$

Since $CPTcreated$ becomes $true$, $\{D\}$ is returned by procedure $extendNetwork$, the set $\mathcal{X}$ is updated, and the procedure $extendNetwork$ is invoked anew. The variable B is (necessarily) chosen. All attempts to create $\mathrm{CPT}(B)$ with an empty or singleton parent set fail. When the procedure $createCPT(B, \{A, D\}, \mathcal{P}, h)$ is invoked, it succeeds and adds the following CPT as $\mathrm{CPT}(B)$ in the CP-net h:

a_1d_1	$b_1 \succ b_2$
a_1d_2	$b_2 \succ b_1$
a_2d_1	$b_2 \succ b_1$
a_2d_2	$b_1 \succ b_2$

The procedure $extendNetwork$ returns $\{B\}$, and in its subsequent call it returns $\emptyset$. Since $\mathcal{X}$ is now empty, the algorithm returns the CP-net h, and terminates.

4 Properties of the Learning Algorithm

We now prove the soundness and completeness of the presented algorithm. For every outcome o over the set $\mathcal{V}$ of variables, and every subset $\mathcal{S} \subseteq \mathcal{V}$, $o_{[\mathcal{S}]}$ denotes the projection of o on $\mathcal{S}$; to avoid clattering, for every $X \in \mathcal{V}$, we write $o_{[X]}$ instead of $o_{[\{X\}]}$. In the outcome $o_{[\mathcal{S}_1]}o_{[\mathcal{S}_2]} \ldots o_{[\mathcal{S}_n]}\mathbf{o}$, with $\mathcal{S}_i \subseteq \mathcal{V}$ and $1 \leq i \leq n$, it holds that $\mathbf{o} \in \mathrm{Asst}(\mathcal{V} \setminus (\mathcal{S}_1 \cup \mathcal{S}_2 \cup \ldots \cup \mathcal{S}_n))$. Finally, for every CP-net N over $\mathcal{V}$, and every $X \in \mathcal{V}$, $\ell_N(X)$ denotes the level of X in N.

Theorem 2 (Soundness of Algorithm $learn$)**.** *If the call $learn(\mathcal{V}, \mathcal{P})$ returns a CP-net h, then h entails every pairwise comparison between outcomes in $\mathcal{P}$.*

Proof (sketch). Let $\mathcal{V}_n = \{X \mid X \in \mathcal{V} \text{ and } \ell_h(X) \leq n\}$. We show that for every $o_i \succ o_j \in \mathcal{P}$, and every positive integer n, if $o_{i[\mathcal{V}_n]} \neq o_{j[\mathcal{V}_n]}$ then $h \models o_{i[\mathcal{V}_n]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}\mathbf{o}$ for every appropriate choice of $\mathbf{o}$. The result follows when $n = |\mathcal{V}|$. Let $\mathcal{V}'_n = \{X \mid X \in \mathcal{V}_n \setminus \mathcal{V}_{n-1} \text{ and } o_{i[X]} \neq o_{j[X]}\}$.

We proceed by induction on n. The base case $n = 1$ follows easily, and we omit the proof. Assume that the claim holds for all values up to some $n \geq 1$, and consider the case of $n + 1$. Assume $o_{i[\mathcal{V}_{n+1}]} \neq o_{j[\mathcal{V}_{n+1}]}$. We only consider the case $\mathcal{V}'_{n+1} \neq \emptyset$. Let $X \in \mathcal{V}'_{n+1}$ and $x_1 = o_{i[X]}$, $x_2 = o_{j[X]}$. Then the CPT of X contains either *(i)* the entry $o_{i[\mathrm{Pa}_h(X)]} : x_1 \succ x_2$, or *(ii)* the entry $o_{j[\mathrm{Pa}_h(X)]} : x_1 \succ x_2$. In case *(i)*, $N \models o_{i[\mathcal{V}_n]}o_{i[X]}\mathbf{o} \succ o_{i[\mathcal{V}_n]}o_{j[X]}\mathbf{o}$. By the inductive hypothesis, $N \models o_{i[\mathcal{V}_n]}o_{j[X]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[X]}\mathbf{o}$ (or $o_{i[\mathcal{V}_n]} = o_{j[\mathcal{V}_n]}$), and by transitivity, $N \models o_{i[\mathcal{V}_n]}o_{i[X]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[X]}\mathbf{o}$. In case *(ii)*, $N \models o_{j[\mathcal{V}_n]}o_{i[X]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[X]}\mathbf{o}$. By the inductive hypothesis, $N \models o_{i[\mathcal{V}_n]}o_{i[X]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{i[X]}\mathbf{o}$ (or $o_{i[\mathcal{V}_n]} = o_{j[\mathcal{V}_n]}$), and by transitivity, $N \models o_{i[\mathcal{V}_n]}o_{i[X]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[X]}\mathbf{o}$. In either case it follows that for every $X \in \mathcal{V}'_{n+1}$, $N \models o_{i[\mathcal{V}_n]}o_{i[X]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[X]}\mathbf{o}$.

Now let $\mathcal{V}^i_{n+1} = \{X^i_1, \ldots, X^i_b\} = \{X \mid X \in \mathcal{V}'_{n+1} \text{ and } \mathrm{CPT}(X) \text{ contains } o_{i[\mathrm{Pa}_h(X)]} : x_1 \succ x_2\}$, and define $\mathcal{V}^j_{n+1}$ analogously. Let $\mathcal{V}^{i,-}_{n+1} = \{X^i_1, \ldots, X^i_{b-1}\}$. It follows that $N \models o_m\mathbf{o} \succ o_{m+1}\mathbf{o}$, for $1 \leq m < b$, where $o_1 = o_{i[\mathcal{V}_n]}o_{i[\mathcal{V}^{i,-}_{n+1}]}$, $o_{b-1} = o_{i[\mathcal{V}_n]}o_{j[\mathcal{V}^{i,-}_{n+1}]}$, and o_m differs from o_{m+1} in the value of exactly one variable $X \in \mathcal{V}^{i,-}_{n+1}$, with $o_{m[X]} = o_{i[X]}$ and $o_{m+1[X]} = o_{j[X]}$. By transitivity, $N \models o_{i[\mathcal{V}_n]}o_{i[\mathcal{V}^{i,-}_{n+1}]}\mathbf{o} \succ o_{i[\mathcal{V}_n]}o_{j[\mathcal{V}^{i,-}_{n+1}]}\mathbf{o}$.

We have already shown that $N \models o_{i[\mathcal{V}_n]}o_{i[X^i_b]}o_{j[\mathcal{V}^{i,-}_{n+1}]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[X^i_b]}o_{j[\mathcal{V}^{i,-}_{n+1}]}\mathbf{o}$. Thus, $N \models o_{i[\mathcal{V}_n]}o_{i[X^i_b]}o_{i[\mathcal{V}^{i,-}_{n+1}]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[X^i_b]}o_{j[\mathcal{V}^{i,-}_{n+1}]}\mathbf{o}$; equivalently $N \models o_{i[\mathcal{V}_n]}o_{i[\mathcal{V}^i_{n+1}]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[\mathcal{V}^i_{n+1}]}\mathbf{o}$. Using similar arguments it holds that $N \models o_{j[\mathcal{V}_n]}o_{j[\mathcal{V}^i_{n+1}]}o_{i[\mathcal{V}^j_{n+1}]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[\mathcal{V}^i_{n+1}]}o_{j[\mathcal{V}^j_{n+1}]}\mathbf{o}$. Hence, $N \models o_{i[\mathcal{V}_n]}o_{i[\mathcal{V}^i_{n+1}]}o_{i[\mathcal{V}^j_{n+1}]}\mathbf{o} \succ o_{j[\mathcal{V}_n]}o_{j[\mathcal{V}^i_{n+1}]}o_{j[\mathcal{V}^j_{n+1}]}\mathbf{o}$, or equivalently $N \models o_{i[\mathcal{V}_{n+1}]}\mathbf{o} \succ o_{j[\mathcal{V}_{n+1}]}\mathbf{o}$, as needed. □

Note that the soundness of the algorithm is unconditional. Given Theorem 1, however, we cannot hope for an unconditional completeness result also. We establish completeness for a special case, that of learning CP-nets that not only entail the comparisons in a set $\mathcal{P}$, but do so in a *transparent* manner.

Definition 5 (Transparent Entailment)**.** *Consider a CP-net N over a set $\mathcal{V}$ of variables that entails every comparison in a set $\mathcal{P}$ of pairwise comparisons between outcomes over $\mathcal{V}$.*

N ***transparently*** *entails* $\mathcal{P}$ *if for every* $\mathbf{o_i}o_{i[\mathrm{Pa}_N(X)]}o_{i[X]} \succ \mathbf{o_j}o_{j[\mathrm{Pa}_N(X)]}o_{j[X]} \in \mathcal{P}$ *with* $o_{i[X]} \neq o_{j[X]}$*,* N *entails either* $\mathbf{o}o_{i[\mathrm{Pa}_N(X)]}o_{i[X]} \succ \mathbf{o}o_{i[\mathrm{Pa}_N(X)]}o_{j[X]}$ *or* $\mathbf{o}o_{j[\mathrm{Pa}_N(X)]}o_{i[X]} \succ \mathbf{o}o_{j[\mathrm{Pa}_N(X)]}o_{j[X]}$ *for some* $\mathbf{o} \in \mathrm{Asst}(\mathcal{V} \setminus (\mathrm{Pa}_N(X) \cup \{X\}))$.

Note that transparency does not restrict the complexity of a CP-net, but indicates how detailed information $\mathcal{P}$ offers about the CP-net. It is easy to show that if $\mathcal{P}$ contains comparisons only between outcomes that differ on exactly one variable (in some sense, the best one can hope for, given Theorem 1), then entailment and transparent entailment coincide. Transparent entailment is, however, a broader notion. Indeed, if $\mathcal{P}$ contains only the preference $a_1b_1c_1 \succ a_2b_2c_2$, then the CP-net N over the variables A, B, C, and with their CPTs containing $: a_1 \succ a_2$, $: b_1 \succ b_2$, and $a_1b_1 : c_1 \succ c_2$ irrespectively of the other entries, transparently entails $\mathcal{P}$. Yet, the preference in $\mathcal{P}$ is between outcomes that differ on three variables. If $\mathcal{P}$ is extended with $a_1b_1c_2 \succ a_1b_1c_1$ and $a_2b_2c_2 \succ a_2b_2c_1$, then no CP-net transparently entails $\mathcal{P}$. Moreover, the algorithm returns failure, although there is a CP-net that entails $\mathcal{P}$.

Theorem 3 (Completeness of Algorithm $learn$)**.** *Consider a set* $\mathcal{P}$ *of pairwise comparisons between outcomes over a set* $\mathcal{V}$ *of variables. If some CP-net* $N \in \mathcal{N}_k$ *over* $\mathcal{V}$ *transparently entails* $\mathcal{P}$*, then the call* $learn(\mathcal{V}, \mathcal{P})$ *returns a CP-net* $h \in \mathcal{N}_k$*.*

Proof (sketch). Consider the Steps *(4)–(13)* of $learn(\mathcal{V}, \mathcal{P})$ during the k-th iteration, and assume by way of contradiction that $\mathcal{X} \neq \emptyset$ at Step *(9)*. Then, the last invocation of $extendNetwork(\mathcal{X}, \mathcal{V}', \mathcal{P}, k, h)$ at Step *(4)* returned $\emptyset$; note that $\mathcal{V}' = \mathcal{V} \setminus \mathcal{X}$. Choose $X \in \mathcal{X}$ such that no variable in $\mathcal{X}$ is a parent variable for X in the CP-net N; since N is acyclic, this can always be done. It follows that all parent variables $\mathrm{Pa}_N(X)$ of X in N are in $\mathcal{V} \setminus \mathcal{X}$. Since $extendNetwork(\mathcal{X}, \mathcal{V}', \mathcal{P}, k, h)$ returned $\emptyset$, it follows that the invocation of $createCPT(X, \mathcal{U}, \mathcal{P}, h)$ failed when $\mathcal{U} = \mathrm{Pa}_N(X)$; a contradiction by the transparency assumption. □

Roughly, transparency implies that the preferences in $\mathcal{P}$ offer information for the dependency of a variable on its parents to be "easily" determined. Thus, backtracking is avoided.

By Theorems 2 and 3, and by observing that $learn(\mathcal{V}, \mathcal{P})$ runs in time $poly(|\mathcal{P}|, |\mathcal{V}|^k)$, we obtain:

Corollary 2 (Consistent-Learning CP-Nets)**.** *The class* $\mathcal{N}_k$ *of CP-nets over a set* $\mathcal{V}$ *of variables is consistent-learnable in time* $poly(|\mathcal{P}|, |\mathcal{V}|^k)$ *given a set* $\mathcal{P}$ *of pairwise comparisons between outcomes over* $\mathcal{V}$ *that is transparently entailed by some* $N \in \mathcal{N}_k$*. In particular, the result holds if all comparisons between outcomes differ on exactly one variable.*

5 Learning with Predictive Guarantees

Beyond simply fitting the available data, the ultimate goal in many real-world domains is to confidently predict a user's preferences on future, and possibly previously unseen, occasions. We examine next a learning setting that builds on the *Probably Approximately Correct* semantics [Valiant, 1984].

A learner observes the user's preferences, modelled as being entailed by some *unknown* CP-net $N \in \mathcal{N}$. To account for the learner's lack of control on what such preferences are obtained, the preferences are assumed to be randomly drawn from some *unknown* probability distribution $\mathcal{D}$.

Definition 6 (Random Instance Oracle)**.** *Given a set* $\mathcal{V}$ *of variables, a probability distribution* $\mathcal{D}$ *over unordered outcome pairs over* $\mathcal{V}$*, and a CP-net* N *over* $\mathcal{V}$*, the* ***(transparent) random instance oracle*** $\mathcal{R}(\mathcal{D}; N)$ *is a procedure that runs in unit time, and on each call* $o_1 \succ o_2 \leftarrow \mathcal{R}(\mathcal{D}; N)$ *returns a comparison* $o_1 \succ o_2$*, where* $\{o_1, o_2\}$ *is drawn randomly and independently from* $\mathcal{D}$*, and* N *(transparently) entails* $o_1 \succ o_2$*.*

Definition 7 (CP-Net PAC-Learnability)**.** *A class* $\mathcal{N}$ *of CP-nets over a set* $\mathcal{V}$ *of variables is* ***PAC-learnable (by*** *a class* $\mathcal{H}$ *of hypotheses) if there exists an algorithm* $\mathcal{L}$ *such that for every probability distribution* $\mathcal{D}$ *over unordered outcome pairs over* $\mathcal{V}$*, every CP-net* $N \in \mathcal{N}$*, and every pair of real numbers* $\delta, \varepsilon \in (0, 1]$*, algorithm* $\mathcal{L}$ *has the following property: given access to* $\mathcal{V}$*,* $\mathcal{R}(\mathcal{D}; N)$*,* δ*, and* ε*, algorithm* $\mathcal{L}$ *runs in time polynomial in* $|\mathcal{V}|$*,* $1/\delta$*,* $1/\varepsilon$*, and* $size(N)$*, and with probability* $1 - \delta$ *returns a hypothesis* h *(in* $\mathcal{H}$*) such that*

$$Pr\left(h \models o_1 \succ o_2 \mid o_1 \succ o_2 \leftarrow \mathcal{R}(\mathcal{D}; N)\right) \geq 1 - \varepsilon.$$

Except with an arbitrarily small probability of failure δ, a learner is expected to return a hypothesis $h \in \mathcal{H}$ that correctly predicts the user's preference $o_1 \succ o_2$ in an arbitrarily high fraction $1 - \varepsilon$ of occasions. An appropriate increase in the allowed resources compensates for this requirement. Although the learner need not return a CP-net as its hypothesis, restricting the learner to do so is possible by fixing $\mathcal{H} = \mathcal{N}$.

We show that learnability under these strong requirements is possible, by observing that the algorithm presented earlier produces *concise* CP-nets $h \in \mathcal{N}$ with maximum in-degree that does not exceed that of the hidden target CP-net $N \in \mathcal{N}$.

Theorem 4 (The Effect of Conciseness)**.** *Consider a class* $\mathcal{N}_k$ *of CP-nets over a set* $\mathcal{V}$ *of variables. If* $\mathcal{N}_k$ *is consistent-learnable by* $\mathcal{N}_k$*, then* $\mathcal{N}_k$ *is PAC-learnable by* $\mathcal{N}_k$*.*

Proof (sketch). The consistent-learnability of $\mathcal{N}_k$ implies an (α, β)-Occam algorithm for $\mathcal{N}_k$ using $\mathcal{N}_k$, for some constant α and $\beta = 0$. The claim follows by standard results [Kearns and Vazirani, 1994, pp. 33–34, Theorem 2.1]. □

An immediate consequence of Theorem 4 is that all hardness results for PAC-learnability that we establish in the remainder of this work also apply to consistent-learnability. On the positive side, Theorem 4 implies:

Corollary 3 (PAC-Learning CP-Nets)**.** *The class* $\mathcal{N}_k$ *of CP-nets over a set* $\mathcal{V}$ *of variables is PAC-learnable by* $\mathcal{N}_k$*, if a transparent random instance oracles is employed, and time* $poly(|\mathcal{V}|^k)$ *is allowed in addition to the time prescribed by Definition 7. (Observe that since* $size(N) \geq 2^k$ *for every* $N \in \mathcal{N}_k$*, then* $poly(|\mathcal{V}|^k) \leq poly(size(N)^{\log |\mathcal{V}|})$*.)*

We emphasize that modulo the restriction imposed on the random instance oracle (corresponding to a restriction on the probability distributions from which comparisons are drawn), and the quasi-polynomial in $|\mathcal{V}|$ running time, Corollary 3 establishes that the presented algorithm is a PAC-learner as per Definition 7. Furthermore, if CP-nets with a constant in-degree are considered, the algorithm runs in polynomial time.

While more efficient algorithms could conceivably exist, we show that this is unlikely, since such algorithms could be used to PAC-learn the class of k-juntas — the class of all boolean functions over n variables that depend on at most k of them, albeit in any arbitrary manner. No PAC-learner for the class of k-juntas is known that runs in time $poly(n)$ for non-constant values of k; the most efficient known one requires time $poly(n^k)$. Improving upon this time would resolve a major long-standing open problem in Computational Learning Theory [Blum, 2003], one, in fact, that was shown to be in the heart of numerous other open problems.

Theorem 5 (Reduction of k-Junta to CP-Net Learning)**.** *Let $\mathcal{J}_k$ be the class of k-juntas over n variables. Consider the class $\mathcal{N}_k$ of all CP-nets over some set $\mathcal{V}$ of $n+1$ variables (and can even assume that all but one variables have zero parents). If $\mathcal{N}_k$ is PAC-learnable (by $\mathcal{N}_k$) in time t, then $\mathcal{J}_k$ is PAC-learnable (by $\mathcal{J}_k$) in time $poly(t, n)$.*

Proof (sketch). Introduce a variable $V_i \in \mathcal{V}$ for each of the n variables x_i over which k-juntas are defined, and an additional variable $Q \in \mathcal{V}$ that corresponds to the output of k-juntas. For each $v \in \{0, 1\}$, let $\overline{v}$ be s.t. $\{v, \overline{v}\} = \{0, 1\}$.

For each k-junta $\varphi \in \mathcal{J}_k$ that depends on the variables $\{x_{i_1}, \ldots, x_{i_k}\}$ let the CP-net $\mathtt{net}(\varphi) \in \mathcal{N}_k$ over $\mathcal{V}$ be s.t.: $\mathtt{net}(\varphi)$ includes links from the variables in $\{V_{i_1}, \ldots, V_{i_k}\}$ to the variable Q, and $\mathrm{CPT}(Q)$ contains for each input $\mathtt{in}$ of φ the entry $\mathtt{in} : \varphi(\mathtt{in}) \succ \overline{\varphi(\mathtt{in})}$. Observe, then, that $\varphi(\mathtt{in}) = \mathtt{out}$ if and only if $\mathtt{net}(\varphi) \models \mathtt{inout} \succ \overline{\mathtt{inout}}$.

The bijection between inputs/outputs of φ to comparisons entailed by $\mathtt{net}(\varphi)$ is computable in time polynomial in n; thus, a PAC-learner for $\mathcal{N}_k$ implies a PAC-learner for $\mathcal{J}_k$.

Finally, if $\mathcal{N}_k$ is PAC-learned by $\mathcal{N}_k$, then $\mathrm{CPT}(Q)$ contains at most 2^k entries, which determine a k-junta in $\mathcal{J}_k$. □

Note, in particular, that the reduction preserves a *properness* property: the PAC-learner returns a hypothesis *in the same* class as the target concept, as opposed to some other hypothesis class $\mathcal{H}$. Recent results imply that properly PAC-learning k-juntas is $\mathtt{W[2]}$-hard [Arvind *et al.*, 2007, Lemma 4], which amounts to an intractability result for obtaining a PAC-learner for k-juntas that runs in time $g(k)poly(n)$, for *any* computable function g. By Theorem 5, we have that:

Corollary 4 (Fixed-Parameter PAC-Learning Hardness)**.** *PAC-learning the CP-net class $\mathcal{N}_k$ by $\mathcal{N}_k$ is $\mathtt{W[2]}$-hard.*

6 Conclusions

The importance of preference learning has been recognized in different scientific areas. This paper presents some of the first results, both positive and negative, concerning the problem of learning CP-nets. On the positive side it introduces a polynomial-time learning algorithm for a restricted class of input examples and CP-nets. Moreover, it establishes that CP-nets that can be learned by this procedure are PAC-learnable. On the negative side it proves that learning is intractable even for rather simple cases. It also shows that properly PAC-learning CP-nets of bounded in-degree is $\mathtt{W[2]}$-hard.

There are many possibilities for future research. In one direction we plan to investigate the integration of our approach with the *global consistency* of [Lang and Mengin, 2008]. An input comparison $a \succ b$ understood under global consistency, requires that the learned CP-net does *not* entail $b \succ a$. Thus, it is possible to learn with both kinds of comparisons, where those under entailment can be seen as *positive* examples, and those under global consistency as *negative* ones. We also intent to study the learnability of other CP-net classes, including the case of CP-nets over non-binary variables, as well the case of the more expressive TCP-nets [Brafman *et al.*, 2006]. In the former case, the algorithm presented herein can be generalized to use a SAT/CSP solver for solving general constraint satisfaction problems. State of the art solvers have been proven very effective in solving real-world problems, and this efficiency can be exploited for learning CP-nets.

Investigating different learning paradigms is another possible line of research. For instance, it would be of practical importance to devise techniques for learning when only partial assignments are available, or when the data does not perfectly fit any hidden CP-net. Finally, online learning is another direction, where the learner is given (possibly adversarially chosen) preferences one by one, and each time makes a prediction on the next preference before obtaining it.

References

[Arvind *et al.*, 2007] V. Arvind, J. Köbler, and W. Lindner. Parameterized learnability of k-juntas and related problems. In *ALT'07*, 2007.

[Aspvall *et al.*, 1979] B. Aspvall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inform. Proc. Letters*, 8, 1979.

[Athienitou and Dimopoulos, 2007] F. Athienitou and Y. Dimopoulos. Learning CP-networks: A preliminary investigation. In *M-Pref'07*, 2007.

[Blum, 2003] A. Blum. Learning a function of r relevant variables (open problem). In *COLT'03*, 2003.

[Boutilier *et al.*, 2003] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of AI Research*, 2003.

[Brafman *et al.*, 2006] R. Brafman, C. Domshlak, and S. Shimony. On graphical modeling of preference and importance. *Journal of AI Research*, 25, 2006.

[Brinker *et al.*, 2006] K. Brinker, J. Fürnkranz, and E. Hüllermeier. A unified model for multilabel classification and ranking. In *ECAI'06*, 2006.

[Cohen *et al.*, 1999] W. Cohen, R. Schapire, and Y. Singer. Learning to order things. *Journal of AI Research*, 10, 1999.

[Kearns and Vazirani, 1994] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, Massachusetts, U.S.A., 1994.

[Lang and Mengin, 2008] J. Lang and J. Mengin. Learning preference relations over combinatorial domains. In *NMR'08*, 2008.

[Valiant, 1984] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.