Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence

Learning Partial Lexicographic Preference Trees over Combinatorial Domains

Xudong Liu and **Miroslaw Truszczynski**
Department of Computer Science
University of Kentucky
Lexington, KY USA
{liu,mirek}@cs.uky.edu

Abstract

We introduce *partial lexicographic preference trees* (PLP-trees) as a formalism for compact representations of preferences over combinatorial domains. Our main results concern the problem of passive learning of PLP-trees. Specifically, for several classes of PLP-trees, we study how to learn (i) a PLP-tree consistent with a dataset of examples, possibly subject to requirements on the size of the tree, and (ii) a PLP-tree correctly ordering as many of the examples as possible in case the dataset of examples is inconsistent. We establish complexity of these problems and, in all cases where the problem is in the class P, propose polynomial time algorithms.

Introduction

Representing and reasoning about preferences are fundamental to decision making and so, of significant interest to artificial intelligence. When the choice is among a few *outcomes* (*alternatives*), representations in terms of explicit enumerations of the preference order are feasible and lend themselves well to formal analysis. However, in many applications outcomes come from *combinatorial domains*. That is, outcomes are described as tuples of values of *issues* (also referred to as *variables* or *attributes*), say $X_1, \ldots, X_p$, with each issue X_i assuming values from some set D_i – its *domain*. Because of the combinatorial size of the space of such outcomes, explicit enumerations of their elements are impractical. Instead, we resort to formalisms supporting intuitive and, ideally, concise *implicit* descriptions of the order. The language of CP-nets (Boutilier et al. 2004) is a prime example of such a formalism.

Recently, however, there has been a rising interest in representing preferences over combinatorial domains by exploiting the notion of the lexicographic ordering. For instance, assuming issues are over the binary domain $\{0, 1\}$, with the preferred value for each issue being 1, a sequence of issues naturally determines an order on outcomes. This idea gave rise to the language of *lexicographic preference models* or *lexicographic strategies*, which has been extensively studied in the literature (Schmitt and Martignon 2006; Dombi, Imreh, and Vincze 2007; Yaman et al. 2008). The formalism of complete *lexicographic preference trees* (LP-trees) (Booth et al. 2010) generalizes the language of lexicographic strategies by arranging issues into decision trees that assign preference ranks to outcomes. An important aspect of LP-trees is that they allow us to model *conditional* preferences on issues and *conditional* ordering of issues. Another formalism, the language of *conditional lexicographic preference trees* (or CLP-trees) (Bräuning and Eyke 2012), extends LP-trees by allowing subsets of issues as labels of nodes.

A central problem in preference representation concerns learning implicit models of preferences (such as lexicographic strategies, LP-trees or CLP-trees), of possibly small sizes, that are consistent with all (or at least possibly many) given examples, each correctly ordering a pair of outcomes. The problem was extensively studied. Booth et al. (2010) considered learning of LP-trees, and Bräuning and Eyke (2012) of CLP-trees.

In this paper, we introduce *partial lexicographic preference trees* (or *PLP-trees*) as means to represent *total preorders* over combinatorial domains. PLP-trees are closely related to LP-trees requiring that every path in the tree contains all issues used to describe outcomes. Consequently, LP-trees describe total orders over the outcomes. PLP-trees relax this requirement and allow paths on which some issues may be missing. Hence, PLP-trees describe total preorders. This seemingly small difference has a significant impact on some of the learning problems. It allows us to seek PLP-trees that minimize the set of issues on their paths, which may lead to more robust models by disregarding issues that have no or little influence on the true preference (pre)order.

The rest of the paper is organized as follows. In the next section, we introduce the language of PLP-trees and describe a classification of PLP-trees according to their complexity. We also define three types of passive learning problems for the setting of PLP-trees. In the following sections, we present algorithms learning PLP-trees of particular types and computational complexity results on the existence of PLP-trees of different types, given size or accuracy. We close with conclusions and a brief account of future work.

Partial Lexicographic Preference Trees

Let $\mathcal{I} = \{X_1, \ldots, X_p\}$ be a set of binary issues, with each X_i having its domain $D_i = \{0_i, 1_i\}$. The corresponding

combinatorial domain is the set $\mathcal{X} = D_1 \times \ldots \times D_p$. Elements of $\mathcal{X}$ are called *outcomes*.

A PLP-tree over $\mathcal{X}$ is binary tree whose every non-leaf node is labeled by an issue from $\mathcal{I}$ and by a preference entry $1 > 0$ or $0 > 1$, and whose every leaf node is denoted by a box $\Box$. Moreover, we require that on every path from the root to a leaf each issue appears *at most* once.

To specify the total preorder on outcomes defined by a PLP-tree T, let us enumerate leaves of T from left to right, assigning them integers $1, 2$, etc. For every outcome α we find its leaf in T by starting at the root of T and proceeding downward. When at a node labeled with an issue X, we descend to the left or to the right child of that node based on the value $\alpha(X)$ of the issue X in α and on the preference assigned to that node. If $\alpha(X)$ is the preferred value, we descend to the left child. We descend to the right child, otherwise. The integer assigned to the leaf that we eventually get to is the *rank* of α in T, written $r_T(\alpha)$. The preorder $\succeq_T$ on distinct outcomes determined by T is defined as follows: $\alpha \succeq_T \beta$ if $r_T(\alpha) \leq r_T(\beta)$ (smaller ranks are "better"). We also define derived relations $\succ_T$ (strict order) and $\approx_T$ (equivalence or indifference): $\alpha \succ_T \beta$ if $\alpha \succeq_T \beta$ and $\beta \not\succeq_T \alpha$, and $\alpha_T \approx_T \beta$ if $\alpha \succeq_T \beta$ and $\beta \succeq_T \alpha$. Clearly, $\succeq_T$ is a total preorder on outcomes partitioning them into strictly ordered clusters of equivalent outcomes.

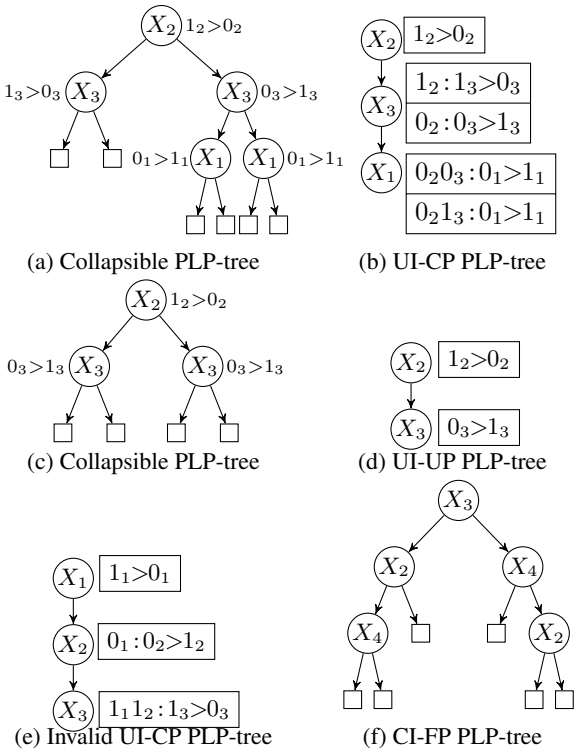
To illustrate the notions just introduced, we consider preference orderings of dinner options over four binary issues. The *appetizer* (X_1) can be either *salad* (0_1) or *soup* (1_1). The *main course* (X_2) is either *beef* (0_2) or *fish* (1_2). The *drink* (X_3) can be *beer* (0_3) or (white) *wine* (1_3). Finally, *dessert* (X_4) can be *ice-cream* (0_4) or *pie* (1_4). An agent could specify her preferences over dinners as a PLP-tree in Figure 1a. The *main course* is the most important issue to the agent and she prefers fish to beef. Her next most important issue is what to *drink* (independently of her selection for the main course). She prefers wine over beer with fish, and beer over wine with beef. If the agent has beef for her main dish, no matter what drink she gets, her next consideration is the *appetizer*, and she prefers salad to soup. In this example, the *dessert* does not figure into preferences at all. The most preferred dinners have fish for the main course and wine for the drink, with all possible combinations of choices for the appetizer and dessert (and so, the cluster of most preferred dinners has four elements).

Classification of PLP-Trees

In the worst case, the size of a PLP-tree is exponential in the number of issues in $\mathcal{I}$. However, some PLP-trees have a special structure that allows us to "collapse" them and obtain more compact representations. This yields a natural classification of PLP-trees, which we describe below.

Let $R \subseteq \mathcal{I}$ be the set of issues that appear in a PLP-tree T. We say that T is *collapsible* if there is a permutation $\hat{R}$ of elements in R such that for every path in T from the root to a leaf, issues that label nodes on that path appear in the same order in which they appear in $\hat{R}$.

If a PLP-tree T is collapsible, we can represent T by a single path of nodes labeled with issues according to the order in which they occur in $\hat{R}$, where a node labeled with an issue X_i is also assigned a *partial conditional preference table* (PCPT) that specifies preferences on X_i, conditioned on values of ancestor issues in the path. These tables make up for the lost structure of T as different ways in which ancestor issues evaluate correspond to different locations in the original tree T. Moreover, missing entries in PCPT of X_i imply equivalence (or indifference) between values of X_i under conditions that do not appear in the PCPT. Clearly, the PLP-tree in Figure 1a is collapsible, and can be represented compactly as a single-path tree with nodes labeled by issues in the permutation and PCPTs (cf. Figure 1b). Such a collapsed path labeled by issues is sometimes denoted as a sequence of issues in $\hat{R}$ connected by $\triangleright$, e.g., $X_2 \triangleright X_3 \triangleright X_1$ for the path in Figure 1b.



(a) Collapsible PLP-tree (b) UI-CP PLP-tree

(c) Collapsible PLP-tree (d) UI-UP PLP-tree

(e) Invalid UI-CP PLP-tree (f) CI-FP PLP-tree

Figure 1: PLP-trees over the dinner domain

Collapsible PLP-trees represented by a single path of nodes will be referred to as *unconditional importance* trees or *UI* trees, for short. The name reflects the fact that the order in which we consider issues when seeking the rank of an outcome is always the same (not conditioned on the values of ancestor issues of higher importance).

Let L be a collapsible PLP-tree. If for every path in L the order of issues labeling the path is exactly $\hat{R}$, and L has the same preference $1 > 0$ on *every* node, then every PCPT in the collapsed tree contains the same preference $1 > 0$, no matter the evaluation of the ancestor issues. Thus, every PCPT in the collapsed form can be simplified to a single *fixed* preference $1 > 0$, a shorthand for its full-sized counterpart. We call the resulting collapsed tree a *UI* tree with *fixed preferences*, or a *UI-FP* PLP-tree.

A similar simplification is possible if every path in L has the same ordering of issues which again is exactly $\hat{R}$, and for every issue X_i all nodes in L labeled with X_i have the same preference on values of X_i (either $1_i > 0_i$ or $0_i > 1_i$). Such collapsed trees are called *UI-UP* PLP-trees, with *UP* standing for *unconditional preference*. As an example, the *UI-UP* tree in Figure 1d is the collapsed representation of the collapsible tree in Figure 1c.

In all other cases, we refer to collapsed PLP-trees as *UI-CP* PLP-trees, with *CP* standing for *conditional preference*. If preferences on an issue in such a tree depend in an essential way on all preceding issues, there is no real saving in the size of representation (instead of an exponential PLP-tree we have a small tree but with preference tables that are of exponential size). However, if the preference on an issue depends only on a few higher importance issues say, never more than one or two (or, more generally, never more than some fixed bound b), the collapsed representation is significantly smaller.

As an aside, we note that not every path of nodes labeled with issues and PCPTs is a *UI* tree. An example is given in Figure 1e. Indeed, one can see that there is no PLP-tree that would collapse to it. There is a simple condition characterizing paths with nodes labeled with issues and PCPTs that are valid *UI* trees. This matter is not essential to our discussion later on and we will not discuss it further here due to the space limit.

When a PLP-tree is not collapsible, the importance of an issue depends on where it is located in the tree. We will refer to such PLP-trees as *conditional importance* trees or *CI* trees.

Let T be a *CI* PLP-tree. We call T a *CI-FP* tree if every non-leaf node in T is labeled by an issue with preference $1 > 0$. An example of a *CI-FP* PLP-tree is shown in Figure 1f, where preferences on each non-leaf node are $1 > 0$ and hence omitted. If, for every issue X_i, all nodes in T labeled with X_i have the same preference ($1_i > 0_i$ or $0_i > 1_i$) on X_i, we say T is a *CI-UP* PLP-tree. All other non-collapsible PLP-trees are called *CI-CP* PLP-trees.

Passive Learning

An *example* is a tuple (α, β, v), where α and β are two *distinct* outcomes from combinatorial domain $\mathcal{X}$ over a set $\mathcal{I} = \{X_1, \ldots, X_p\}$ of binary issues, and $v \in \{0, 1\}$. An example $(\alpha, \beta, 1)$ states that α is strictly preferred to β ($\alpha \succ \beta$). Similarly, an example $(\alpha, \beta, 0)$ states that α and β are equivalent ($\alpha \approx \beta$). Let $\mathcal{E} = \{e_1, \ldots, e_m\}$ be a set of examples over $\mathcal{I}$, with $e_i = (\alpha_i, \beta_i, v_i)$. We set $\mathcal{E}^{\approx} = \{e_i \in \mathcal{E} : v_i = 0\}$, and $\mathcal{E}^{\succ} = \{e_i \in \mathcal{E} : v_i = 1\}$. In the following, we denote by p and m the number of issues and the number of examples, respectively.

For a PLP-tree T in full representation we denote by $|T|$ the size of T, that is, the number of nodes in T. If T stands for a *UI* tree, we write $|T|$ for the size of T measured by the total size of preference tables associated with issues in T. The size of a preference table is the total size of preferences in it, each preference measured as the number of values in the condition plus 1 for the preferred value in the domain of the issue. In particular, the sizes of *UI-FP* and *UI-UP* trees are given by the number of nodes on the path.

A PLP-tree T *satisfies* an example e if T orders the two outcomes of e in the same way as they are ordered in e. Otherwise, T *falsifies* e. Formally, T *satisfies* $e = (\alpha, \beta, 1)$ if $\alpha \succ_T \beta$, and T *satisfies* $e = (\alpha, \beta, 0)$ if $\alpha \approx_T \beta$. We say T is *consistent* with a set $\mathcal{E}$ of examples if T satisfies every example in $\mathcal{E}$.

In this work, we study the following passive learning problems for PLP-trees of all types we introduced.

Definition 1. Consistent-learning (CONSLEARN): given an example set $\mathcal{E}$, decide whether there exists a PLP-tree T (of a particular type) such that T is consistent with $\mathcal{E}$.

Definition 2. Small-learning (SMALLLEARN): given an example set $\mathcal{E}$ and a positive integer l ($l \leq |\mathcal{E}|$), decide whether there exists a PLP-tree T (of a particular type) such that T is consistent with $\mathcal{E}$ and $|T| \leq l$.

Definition 3. Maximal-learning (MAXLEARN): given an example set $\mathcal{E}$ and a positive integer k ($k \leq m$), decide whether there exists a PLP-tree T (of a particular type) such that T satisfies at least k examples in $\mathcal{E}$.

Learning UI PLP-trees

In this section, we study the passive learning problems for collapsible PLP-trees in their collapsed representations as *UI-FP*, *UI-UP* and *UI-CP* trees.

The CONSLEARN Problem

The CONSLEARN problem is in the class P for *UI-FP* and *UI-UP* trees. To show it, we present a general template of an algorithm that learns a *UI* tree. Next, for each of the classes *UI-FP* and *UI-UP*, we specialize the template to a polynomial-time algorithm.

The template algorithm is shown as Algorithm 1. The input consists of a set $\mathcal{E}$ of examples and a set $\mathcal{I}$ of issues from which node labels can be selected. Throughout the execution, the algorithm maintains a set S of unused issues, initialized to $\mathcal{I}$, and a set of examples that are not yet ordered by the tree constructed so far.

If the set of strict examples is empty, the algorithm returns an empty tree. Otherwise, the algorithm identifies the set $AI(\mathcal{E}, S)$ of issues in S that are *available* for selection as the label for the next node. If that set is empty, the algorithm terminates with failure. If not, an issue, say X_l, is selected from $AI(\mathcal{E}, S)$, and a PCPT for that issue is constructed. Then the sets of examples not ordered yet and of issues not used yet are updated, and the steps repeat.

To obtain a learning algorithm for a particular class of *UI* trees (*UI-FP* or *UI-UP*) we need to specify the notion of an available issue (needed for line 3) and describe how to construct a partial conditional preference table (needed for line 8).

To this end, let us define $NEQ(\mathcal{E}, S)$ to be the set of all issues in S (where $S \subseteq \mathcal{I}$) that incorrectly handle at least one equivalent example in $\mathcal{E}^{\approx}$. That is, for an issue $X \in S$ we have $X \in NEQ(\mathcal{E}, S)$ precisely when for some example $(\alpha, \beta, 0)$ in $\mathcal{E}$, $\alpha(X) \neq \beta(X)$. Similarly, let us define $EQ(\mathcal{E}, S)$ to be the set of issues in S that do not order any

Algorithm 1: Procedure *learnUI* that learns a UI tree

Input: $\mathcal{E}$ and $S = \mathcal{I}$
Output: A sequence T of issues from $\mathcal{I}$ and PCPTs that define a *UI* tree consistent with $\mathcal{E}$, or FAILURE if such a tree does not exist

1 $T \leftarrow$ empty sequence;
2 **while** $\mathcal{E}^{\succ} \neq \emptyset$ **do**
3 Construct $AI(\mathcal{E}, S)$;
4 **if** $AI(\mathcal{E}, S) = \emptyset$ **then**
5 **return** FAILURE;
6 **end**
7 $X_l \leftarrow$ an element from $AI(\mathcal{E}, S)$;
8 Construct *PCPT*(X_l);
9 $T \leftarrow T, (X_l, \textit{PCPT}(X_l))$;
10 $\mathcal{E} \leftarrow \mathcal{E} \backslash \{e \in \mathcal{E}^{\succ} : e \text{ is decided on } X_l\}$;
11 $S \leftarrow S \backslash \{X_l\}$;
12 **end**
13 **return** T;

of the strict examples in $\mathcal{E}$. That is, for an issue $X \in S$ we have $X \in EQ(\mathcal{E}, S)$ precisely when for every example $(\alpha, \beta, 1)$ in $\mathcal{E}$, $\alpha(X) = \beta(X)$.

Fixed Preferences. For the problem of learning *UI-FP* trees, we define $AI(\mathcal{E}, S)$ to contain every issue $X \notin NEQ(\mathcal{E}, S)$ such that
(1) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$.

Proposition 1. *If there is a UI-FP tree consistent with all examples in $\mathcal{E}$ and using only issues from S as labels, then an issue $X \in S$ is a top node of some such tree if and only if $X \in AI(\mathcal{E}, S)$.*

Proof. Let T be a *UI* tree consistent with $\mathcal{E}$ and having only issues from S as labels. Let X be the issue labeling the top node of T. Clearly, $X \notin NEQ(\mathcal{E}, S)$, as otherwise, T would strictly order two outcomes α and β such that $(\alpha, \beta, 0) \in \mathcal{E}^{\approx}$. To prove condition (1), let us consider any example $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$. Since T is consistent with $(\alpha, \beta, 1)$, $\alpha(X) \geq \beta(X)$. Consequently, $X \in AI(\mathcal{E}, S)$.

Conversely, let $X \in AI(\mathcal{E}, S)$ and let T be a *UI-FP* tree consistent with all examples in $\mathcal{E}$ and using only issues from S as labels (such a tree exists by assumption). If X labels the top node in T, we are done. Otherwise, let T' be a tree obtained from T by adding at the top of T another node, labeling it with X and removing from T the node labeled by X, if such a node exists. By the definition of $AI(\mathcal{E}, S)$ we have that $X \notin NEQ(\mathcal{E}, S)$ and that condition (1) holds for X. Using these properties, one can show that T' is also a *UI-FP* tree consistent with all examples in $\mathcal{E}$. Since the top node of T' is labeled by X, the assertion follows. □

We now specialize Algorithm 1 by using in line 3 the definition of $AI(\mathcal{E}, S)$ given above and by setting each *PCPT*(X_l) to the fixed unconditional preference $1_l > 0_l$. Proposition 1 directly implies the correctness of this version of Algorithm 1.

Theorem 2. *Let $\mathcal{E}$ be a set of examples over a set $\mathcal{I}$ of binary issues. Algorithm 1 adjusted as described above terminates and outputs a sequence T representing a UI-FP tree consistent with $\mathcal{E}$ if and only if such a tree exists.*

We note that issues in $NEQ(\mathcal{E}, S)$ are never used when constructing $AI(\mathcal{E}, S)$. Thus, in the case of *UI-FP* trees, S could be initialized to $\mathcal{I} \backslash NEQ(\mathcal{E}, \mathcal{I})$. In addition, if an issue selected for the label of the top node belongs to $EQ(\mathcal{E}^{\succ}, S)$, it does not in fact decide any of the strict examples in $\mathcal{E}$ and can be dropped. The resulting tree is also consistent with all the examples. Thus, the definition of $AI(\mathcal{E}, S)$ can be refined by requiring one more condition: $X \notin EQ(\mathcal{E}^{\succ}, S)$. That change does not affect the correctness of the algorithm but eliminates a possibility of generating trees with "redundant" levels.

Unconditional Preferences. The case of learning *UI-UP* trees is very similar to the previous one. Specifically, we define $AI(\mathcal{E}, S)$ to contain an issue $X \in S$ precisely when $X \notin NEQ(\mathcal{E}, S)$ and
(2) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$, or for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \leq \beta(X)$.

We obtain an algorithm learning *UI-UP* trees by using in line 3 the present definition of $AI(\mathcal{E}, S)$. In line 8, we take for *PCPT*(X_l) either $1_l > 0_l$ or $0_l > 1_l$ (depending on which of the two cases in (2) holds for X_l).

The correctness of this algorithm follows from a property similar to that in Proposition 1.

As in the previous case, here too S could be initialized to $\mathcal{I} \setminus NEQ$, and the condition $X \notin EQ(\mathcal{E}^{\succ}, S)$ could be added to the definition of $AI(\mathcal{E}, S)$.

Conditional Preferences. The problem is in NP because one can show that if a *UI-CP* tree consistent with $\mathcal{E}$ exists (*a priori*, it does not have to have size polynomial in the size of $\mathcal{E}$), then another such tree of size polynomial in the size of $\mathcal{E}$ exists, as well. We conjecture that the general problem of learning *UI-CP* trees is, in fact, NP-complete. As we have only partial results for this case, the study of the *UI-CP* tree learning will be the subject of future work.

The SMALLLEARN Problem

Algorithm 1 produces a *UI* PLP-tree consistent with $\mathcal{E}$, if one exists. In many cases, it is desirable to compute a small, sometimes even the smallest, representation consistent with $\mathcal{E}$. We show that these problems for *UI* trees are NP-hard.

Theorem 3. *The* SMALLLEARN *problem is NP-complete for each class of $\{UI\} \times \{FP, UP, CP\}$.*

Proof. We present the proof only in the case of *UI-FP*. The argument in other cases (*UI-UP* and *UI-CP*) is similar.

(Membership) One can guess a *UI-FP* PLP-tree T in linear time, and verify in polynomial time that T has at most l issues and satisfies every example in $\mathcal{E}$.

(Hardness) We present a polynomial-time reduction from the *hitting set problem* (HSP), which is NP-complete (Garey and Johnson 1979). To recall, in HSP we are given a finite set $U = \{a_1, \ldots, a_n\}$, a collection $C = \{S_1, \ldots, S_d\}$ of subsets of U with $\bigcup_{S_i \in C} S_i = U$, and a positive integer $k \leq n$, and the problem is to decide whether U has a hitting set U' such that $|U'| \leq k$ ($U' \subseteq U$ is a *hitting* set for C if $U' \cap S_i \neq \emptyset$ for all $S_i \in C$). Given an instance of HSP, we construct an instance of our problem as follows.
1. $\mathcal{I} = \{X_i : a_i \in U\}$ (thus, $p = n$).

2. $\mathcal{E} = \{(\mathbf{s}_i, \mathbf{0}, 1) : S_i \in C\}$, where $\mathbf{s}_i$ is a p-bit vector such that $\mathbf{s}_i[j] = 1 \Leftrightarrow a_j \in S_i$ and $\mathbf{s}_i[j] = 0 \Leftrightarrow a_j \notin S_i$ $(1 \leq j \leq p)$, and $\mathbf{0}$ is a p-bit vector of all 0's (thus, $m = d$).
3. We set $l = k$.

We need to show that U has a hitting set of size at most k if and only if there exists a *UI-FP* PLP-tree of size at most l consistent with $\mathcal{E}$.

($\Rightarrow$) Assume U has a hitting set U' of size k. Let U' be $\{a_{j_1}, \ldots, a_{j_k}\}$. Define a *UI-FP* PLP-tree $L = X_{j_1} \triangleright \ldots \triangleright X_{j_k}$. We show that L is consistent with $\mathcal{E}$. Let $e = (\alpha_e, \beta_e, 1)$ be an arbitrary example in $\mathcal{E}$, where $\alpha_e = \mathbf{s}_i$ and $\beta_e = \mathbf{0}$. Since U' is a hitting set, there exists r, $1 \leq r \leq k$, such that $a_{j_r} \in S_i$. Thus, there exists r, $1 \leq r \leq k$, such that $\alpha_e(X_{j_r}) = 1$. Let r be the smallest with this property. It is clear that e is decided at X_{j_r}; thus, we have $\alpha_e \succ_L \beta_e$.

($\Leftarrow$) Assume there is a *UI-FP* PLP-tree L of l issues in $\mathcal{I}$ such that L is consistent with $\mathcal{E}$. Moreover, we assume $L = X_{j_1} \triangleright \ldots \triangleright X_{j_l}$. Let $U' = \{a_{j_1}, \ldots, a_{j_l}\}$. We show by means of contradiction. Assume that U' is not a hitting set. That is, there exists a set $S_i \in C$ such that $U' \cap S_i = \emptyset$. Then, there exists an example $e = (\alpha_e, \beta_e, 1)$, where $\alpha_e = \mathbf{s}_i$ and $\beta_e = \mathbf{0}$, such that $\alpha_e \approx_L \beta_e$ because none of the issues $\{X_i : \alpha_e(X_i) = 1\}$ show up in L. This is a contradiction! Thus, U' is a hitting set. □

Corollary 4. *Given a set $\mathcal{E}$ of examples $\{e_1, \ldots, e_m\}$ over $\mathcal{I} = \{X_1, \ldots, X_p\}$, finding the smallest PLP-tree in each class of $\{UI\} \times \{FP, UP, CP\}$ consistent with $\mathcal{E}$ is NP-hard.*

Consequently, it is important to study fast heuristics that aim at approximating trees of optimal size. Here, we propose a greedy heuristic for Algorithm 1. In every iteration the heuristic selects the issue $X_l \in AI(\mathcal{E}, S)$ that decides the most examples in $\mathcal{E}^{\succ}$. However, for some dataset the resulting greedy algorithm does not perform well: the ratio of the size of the tree computed by our algorithm to the size of the optimal sequence may be as large as $\Omega(p)$. To see this, we consider the following input.

$(1_1 0_2 0_3 0_4, 0_1 0_2 0_3 0_4, 1)$
$(1_1 1_2 0_3 0_4, 0_1 0_2 0_3 0_4, 1)$
$(1_1 0_2 1_3 0_4, 0_1 0_2 0_3 0_4, 1)$
$(0_1 0_2 0_3 1_4, 1_1 0_2 0_3 0_4, 1)$

For each class of $\{UI\} \times \{FP, UP\}$, Algorithm 1 in the worst case computes $X_2 \triangleright X_3 \triangleright X_4 \triangleright X_1$, whereas the optimal tree is $X_4 \triangleright X_1$ (with the PCPTs omitted as they contain only one preference and so, they do not change the asymptotic size of the tree). This example generalizes to the arbitrary number p of issues. Thus, the greedy algorithm to learn small *UI* trees is no better than any other algorithm in the worst case.

Approximating HSP has been extensively studied over the last decades. It has been shown (Lund and Yannakakis 1994) that, unless $NP \subset DTIME(n^{poly \log n})$, HSP cannot be approximated in polynomial time within factor of $c \log n$, where $0 < c < \frac{1}{4}$ and n is the number of elements in the input. The reduction we used above shows that this result carries over to our problem.

Theorem 5. *Unless $NP \subset DTIME(n^{poly \log n})$, the problem of finding the smallest PLP-tree in each class of $\{UI\} \times \{FP, UP, CP\}$ consistent with $\mathcal{E}$ cannot be approximated in polynomial time within factor of $c \log p$, where $0 < c < \frac{1}{4}$.*

It is an open problem whether this result can be strengthened to a factor linear in p (cf. the example for the worst-case behavior of our simple greedy heuristic).

The MAXLEARN Problem

When there is no *UI* PLP-tree consistent with the set of all examples, it may be useful to learn a *UI* PLP-tree satisfying as many examples as possible. We show this problem is in fact NP-hard for all three classes of *UI* trees.

Theorem 6. *The* MAXLEARN *problem is NP-complete for each class of $\{UI\} \times \{FP, UP, CP\}$.*

Sketch. The problem is in NP. This is evident for the case of *UI-FP* and *UI-UP* trees. If $\mathcal{E}$ is a given set of examples, and k a required lower bound on the number of examples that are to be correctly ordered, then witness trees in these classes (trees that correctly order at least k examples in $\mathcal{E}$) have size polynomial in the size of $\mathcal{E}$. Thus, verification can be performed in polynomial time. For the case of *UI-CP* trees, one can show that if there is a *UI-CP* tree correctly ordering at least k examples in $\mathcal{E}$, then there exists such tree of size polynomial in $|\mathcal{E}|$.

The hardness part follows from the proof in the setting of learning lexicographic strategies (Schmitt and Martignon 2006), adapted to the case of *UI* PLP-trees. □

Corollary 7. *Given a set $\mathcal{E}$ of examples $\{e_1, \ldots, e_m\}$ over $\mathcal{I} = \{X_1, \ldots, X_p\}$, finding a PLP-tree in each class of $\{UI\} \times \{FP, UP, CP\}$ satisfying the maximum number of examples in $\mathcal{E}$ is NP-hard.*

Learning CI PLP-trees

Finally, we present results on the passive learning problems for PLP-trees in classes $\{CI\} \times \{FP, UP, CP\}$. We recall that these trees assume full (non-collapsed) representation.

The CONSLEARN Problem

We first show that the CONSLEARN problem for class *CI-UP* is NP-complete. We then propose polynomial-time algorithms to solve the CONSLEARN problem for the classes *CI-FP* and *CI-CP*.

Theorem 8. *The* CONSLEARN *problem is NP-complete for class CI-UP.*

Sketch. The problem is in NP because the size of a witness, a *CI-UP* PLP-tree consistent with $\mathcal{E}$, is bounded by $|\mathcal{E}|$ (one can show that if a *CI-UP* tree consistent with $\mathcal{E}$ exists, then it can be modified to a tree of size no larger than $O(|\mathcal{E}|)$). Hardness follows from the proof by Booth et al. (2010) showing CONSLEARN is NP-hard in the setting of LP-trees. □

For the two other classes of trees, the problem is in P. This is demonstrated by polynomial-time Algorithm 2 adjusted for both classes.

Fixed Preference. For class *CI-FP*, we define $AI(\mathcal{E}, S)$ to contain issue $X \notin NEQ(\mathcal{E}, S)$ if
(3) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$.

Algorithm 2: The recursive procedure *learnCI* that learns a CI PLP-tree

Input: $\mathcal{E}$, $S = \mathcal{I}$, and t: an unlabeled node
Output: A CI PLP-tree over S consistent with $\mathcal{E}$, or FAILURE

1 **if** $\mathcal{E}^{\succ} = \emptyset$ **then**
2 | Label t as a leaf and **return**;
3 **end**
4 Construct $AI(\mathcal{E}, S)$;
5 **if** $AI(\mathcal{E}, S) = \emptyset$ **then**
6 | **return** FAILURE and terminate;
7 **end**
8 Label t with tuple (X_l, x_l) where X_l is from $AI(\mathcal{E}, S)$, and x_l is the preferred value on X_l;
9 $\mathcal{E} \leftarrow \mathcal{E} \backslash \{e \in \mathcal{E}^{\succ} : e \text{ is decided on } X_l\}$;
10 $S \leftarrow S \backslash \{X_l\}$;
11 Create two edges u_l, u_r and two unlabeled nodes t_l, t_r such that $u_l = \langle t, t_l \rangle$ and $u_r = \langle t, t_r \rangle$;
12 $\mathcal{E}_l \leftarrow \{e \in \mathcal{E} : \alpha_e(X_j) = \beta_e(X_j) = x_l\}$;
13 $\mathcal{E}_r \leftarrow \{e \in \mathcal{E} : \alpha_e(X_j) = \beta_e(X_j) = \overline{x_l}\}$;
14 $learnCI(\mathcal{E}_l, S, t_l)$;
15 $learnCI(\mathcal{E}_r, S, t_r)$;

Proposition 9. *If there is a CI-FP tree consistent with all examples in $\mathcal{E}$ and using only issues from S as labels, then an issue $X \in S$ is a top node of some such tree if and only if $X \in AI(\mathcal{E}, S)$.*

Proof. It is clear that if there exists a *CI-FP* PLP-tree consistent with $\mathcal{E}$ and only using issues from S as labels, then the fact that $X \in S$ labels the root of some such tree implies $X \in AI(\mathcal{E}, S)$.

Now we show the other direction. Let T be the *CI-FP* tree over a subset of S consistent with $\mathcal{E}$, X be an issue such that $X \in AI(\mathcal{E}, S)$. If X is the root issue in T, we are done. Otherwise, we construct a *CI-FP* tree T' by creating a root, labeling it with X, and make one copy of T the left subtree of T' (T'_l) and another, the right subtree of T' (T'_r). For a node t and a subtree B in T, we write t'_l and B'_l, respectively, for the corresponding node and subtree in T'_l. We define t'_r and B'_r similarly. If X does not appear in T, we are done constructing T'; otherwise, we update T' as follows.

1). For every node $t \in T$ labeled by X such that t has two leaf children, we replace the subtrees rooted at t'_l and t'_r in T'_l and T'_r with leaves.

2). For every node $t \in T$ labeled by X such that t has one leaf child and a non-leaf subtree B, we replace the subtree rooted at t'_l in T'_l with B'_l, and the subtree rooted at t'_r in T'_r with a leaf, if $t \in T$ has a right leaf child; otherwise, we replace the subtree rooted at t'_l in T'_l with a leaf, and the subtree rooted at t'_r in T'_r with B'_r.

3). Every other node $t \in T$ labeled by X has two non-leaf subtrees: left non-leaf subtree BL and right BR. For every such node $t \in T$, we replace the subtree rooted at t'_l in T'_l with BL'_l, and the subtree rooted at t'_r in T'_r with BR'_r.

One can show that this construction results in a *CI-CP* tree consistent with $\mathcal{E}$ and, clearly, it has its root labeled with X. Thus, the assertion follows. □

Proposition 9 clearly implies the correctness of Algorithm 2 with $AI(\mathcal{E}, S)$ defined as above for class *CI-FP* and each $x_l \in (X_l, x_l)$ set to 1.

Theorem 10. *Let $\mathcal{E}$ be a set of examples over a set $\mathcal{I}$ of binary issues. Algorithm 2 adjusted as described above terminates and outputs a CI-FP tree T consistent with $\mathcal{E}$ if and only if such a tree exists.*

Conditional Preference. For class *CI-CP*, we define that $AI(\mathcal{E}, S)$ contains issue $X \notin NEQ(\mathcal{E})$ if
(4) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$, or for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \leq \beta(X)$.

We obtain an algorithm learning *CI-CP* trees by using in line 4 the present definition of $AI(\mathcal{E}, S)$. In line 8, we take for x_l either 1 or 0 (depending on which of the two cases in (4) holds for X_l). The correctness of this algorithm follows from a property similar to that in Proposition 9.

The SMALLLEARN and MAXLEARN Problems

Due to space limits, we only outline the results we have for this case. Both problems for the three *CI* classes are NP-complete. They are in NP since if a witness PLP-tree exists, one can modify it so that its size does not exceed the size of the input. Hardness of the SMALLLEARN problem for *CI* classes follows from the proof of Theorem 3, whereas the hardness of the MAXLEARN problem for *CI* cases follows from the proof by Schmitt and Martignon (2006).

Conclusions and Future Work

We proposed a preference language, *partial lexicographic preference trees*, *PLP-trees*, as a way to represent preferences over combinatorial domains. For several natural classes of PLP-trees, we studied passive learning problems: CONSLEARN, SMALLLEARN and MAXLEARN. All complexity results we obtained are summarized in tables in Figure 2. The CONSLEARN problem for *UI-CP* trees is as of now unsettled. While we are aware of subclasses of *UI-CP* trees for which polynomial-time algorithms are possible, we conjecture that in general, the problem is NP-complete.

	FP	UP	CP
UI	P	P	*NP*
CI	P	NPC	P

(a) CONSLEARN

	FP	UP	CP
UI	NPC	NPC	NPC
CI	NPC	NPC	NPC

(b) SMALLLEARN & MAXLEARN

Figure 2: Complexity results for passive learning problems

For the future research, we will develop good heuristics for our learning algorithms. We will implement these algorithms handling issues of, in general, finite domains of values, and evaluate them on both synthetic and real-world preferential datasets. With PLP-trees of various classes learned, we will compare our models with the ones learned through other learning approaches on predicting new preferences.

References

Booth, R.; Chevaleyre, Y.; Lang, J.; Mengin, J.; and Sombattheera, C. 2010. Learning conditionally lexicographic preference relations. In *ECAI*, 269–274.

Boutilier, C.; Brafman, R.; Domshlak, C.; Hoos, H.; and Poole, D. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research* 21:135–191.

Bräuning, M., and Eyke, H. 2012. Learning conditional lexicographic preference trees. *Preference learning: problems and applications in AI*.

Dombi, J.; Imreh, C.; and Vincze, N. 2007. Learning lexicographic orders. *European Journal of Operational Research* 183:748–756.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Lund, C., and Yannakakis, M. 1994. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)* 41(5):960–981.

Schmitt, M., and Martignon, L. 2006. On the complexity of learning lexicographic strategies. *The Journal of Machine Learning Research* 7:55–83.

Yaman, F.; Walsh, T. J.; Littman, M. L.; and Desjardins, M. 2008. Democratic approximation of lexicographic preference models. In *Proceedings of the 25th international conference on Machine learning*, 1200–1207. ACM.