EPiC Series in Computing

Volume 41, 2016, Pages 314–328

GCAI 2016. 2nd Global Conference on Artificial Intelligence

Learning Partial Lexicographic Preference Trees and Forests over Multi-Valued Attributes

Xudong Liu[1,2]*and Miroslaw Truszczynski[1]

[1] University of Kentucky, USA
{liu,mirek}@cs.uky.edu
[2] University of North Florida, USA
xudong.liu@unf.edu

Abstract

Partial lexicographic preference trees, or *PLP-trees*, form an intuitive formalism for compact representation of qualitative preferences over combinatorial domains. We show that PLP-trees can be used to accurately model preferences arising in practical situations, and that high-accuracy PLP-trees can be effectively learned. We also propose and study learning methods for a variant of our model based on the concept of a *PLP-forest*, a *collection* of PLP-trees, where the preference order specified by a PLP-forest is obtained by aggregating the orders of its constituent PLP-trees. Our results demonstrate the potential of both approaches, with learning PLP-forests showing particularly promising behavior.

1 Introduction

Preferences are ubiquitous in decision making. They have been extensively studied by researchers in artificial intelligence, psychology, operations research, and social choice theory. Learning preference models, that is, expressions concisely representing a preference order has been central to this research. Much of the attention was focused on learning utility functions that represent preference orders quantitatively [7].

Recently, researchers proposed several *qualitative* models of preference orders arguing that they are more directly aligned with conventions humans use when expressing their preferences. They include *conditional preference networks* (CP-nets) [2], and models ordering outcomes lexicographically such as lexicographic strategies [16], conditional preference trees (CP-trees) [17], lexicographic preference trees (LP-trees) [1], conditional lexicographic preference trees [3], partial lexicographic preference trees (PLP-trees) [12], and preference trees [6, 13]. As with quantitative models, learning qualitative models is important because eliciting them directly from users is often impractical. However, while learning CP-nets has received a fair amount of attention [11, 5, 10, 9], the study of learning lexicographic models is still in the early stages. The results obtained so far concern mostly learning LP-trees [1] and conditional lexicographic preference trees [3]. Other models received less attention. In particular, no algorithms for

*The results of this paper were obtained when Xudong Liu was a Ph.D. student at the University of Kentucky. The final version of the paper was prepared after he joined the University of North Florida – his current affiliation.

learning PLP-trees have yet been proposed even though PLP-trees retain the simplicity of LP-trees but also offer flexibility that makes them less sensitive to overfitting.

In this work, we study the problem of learning PLP-trees over *multi-valued* attributes, which generalize the original binary version of PLP-trees [12]. As in the binary case, multi-valued PLP-trees can be grouped into four classes that capture different types of orders: *unconditional importance and unconditional preference* (UIUP) trees, *unconditional importance and conditional preference* (UICP) trees, *conditional importance and unconditional preference* (CIUP) trees, and *conditional importance and conditional preference* (CICP) trees.

The computational complexity of learning PLP-trees is well understood. Schmitt and Martignon [16] showed that for lexicographic strategies, a special case of UIUP PLP-trees, computing a lexicographic strategy maximizing the number of correctly handled examples is NP-hard, and Liu and Truszczynski [12] extended this result to other classes of PLP-trees. Therefore, in this paper, we focus on the problem of practicality of learning PLP-trees and using them to represent preferences. To this end, we introduce several best-agreement and approximate algorithms to learn PLP-trees (of the four types above). We show experimentally that they are effective on several domains and datasets, and generate trees that accurately approximate the preference order being modeled. To support our experiments, following Bräuning and Hüllermeier [3], we generated a library of datasets of preference examples deriving them from datasets of examples developed by the machine learning community to support research on the classification problem.

When learning a decision tree, a problem that may arise is overfitting. To reduce its effect, Breiman [4] proposed learning a *random forest*, that is, a set of uncorrelated decision trees learned from randomly selected sets of examples. The random forest learning algorithm [4] first generates several decision trees (a forest), randomizing the attributes used in their construction. To classify an instance, the algorithm *aggregates* the predictions made by the trees in the forest using the majority rule. We adapted that approach to the setting of PLP-trees. A *PLP-forest* is a collection of PLP-trees. PLP-trees in a PLP-forest are learned using randomly selected *small* fragments of a training set. To predict if one outcome is preferred over another, we apply the *pairwise majority rule* (PMR), a simple and effective voting rule studied in social choice. We adjust algorithms learning PLP-trees to the setting of PLP-forests and study their effectiveness both in terms of time and accuracy.

The key findings supported by our results are: (1) PLP-trees and PLP-forests are expressive preference models. Experiments with the datasets we constructed from commonly used machine learning classification domains showed that the accuracy of learned models typically exceeded 85%, often exceeded 90%, and in some cases was as high as 95%. (2) PLP-forests aggregated by PMR provide in general higher accuracy than PLP-trees. (3) PLP-trees and PLP-forests learned by a greedy approximation method have accuracy comparable to *best-agreement* PLP-trees and PLP-forests learned by maximizing the number of correctly handled examples in the training set. Moreover, because of overfitting arising in "best-agreement" trees and forests, in some cases, heuristic approaches offer an even better accuracy. (4) Approximation learning methods are fast and can work with large datasets; methods based on learning best-agreement trees can also be effective in practice, especially when we learn PLP-forests, where we bound the number of examples each tree in the forest is learned from.

2 Partial Lexicographic Preference Trees

PLP-trees were originally defined to represent preference orders over combinatorial domains with *binary* attributes [12]. In this work, we expand the definition to the general case where attributes are multi-valued (not necessarily binary), which is relevant to practical applications.

Let $\mathcal{A} = \{X_1, \ldots, X_p\}$ be a set of attributes, with each X_i having a domain D_i, with the sizes of all domains bounded by a constant. The corresponding *combinatorial domain* over $\mathcal{A}$ is the Cartesian product $CD(\mathcal{A}) = D_1 \times \ldots \times D_p$. Elements in $CD(\mathcal{A})$ are called *outcomes*.

A PLP-tree over $CD(\mathcal{A})$ is an ordered labeled tree, where: (1) every non-leaf node is labeled by some attribute from $\mathcal{A}$, say X_i, and by a *local preference* $>_i$, a total strict order on the corresponding domain D_i; (2) every non-leaf node labeled by an attribute X_i has $|D_i|$ outgoing edges; (3) every leaf node is denoted by $\Box$; and (4) on every path from the root to a leaf each attribute appears *at most once* as a label.

Each outcome $\alpha \in CD(\mathcal{A})$ determines in a PLP-tree T its *outcome path*, $P(\alpha, T)$. It starts at the root of T and proceeds downward. When at a node n labeled with an attribute X, the path descends to the next level based on the value $\alpha(X)$ of the attribute X in the outcome α and on the local preference order associated with n. Namely, if $\alpha(X)$ is the i-th most preferred value in this order, the path descends to the i-th child of n.

The *shared path segment* of α and β in T is the longest common prefix of both $P(\alpha, T)$ and $P(\beta, T)$. If the last node on that segment is not a leaf, we call this node the *divergence node* for α and β. We say that outcome α is at least as good as β ($\alpha_T \succeq_T \beta$) if (1) the divergence node is not defined (the last node on the shared path segment is a leaf, and so both outcomes end up in the same cluster of equivalent outcomes), or (2) the divergence node is defined and $\alpha(X_{Div}) >_{Div} \beta(X_{Div})$, where X_{Div} is the attribute labeling the divergence node and $>_{Div}$ is the local preference order on the domain of X_{Div}. Clearly, the condition (1) defines the associated equivalence relation $\approx_T$ and the condition (2) defines the associated strict order relation $\succ_T$.

To illustrate, let us consider the domain of cars described by four multi-valued attributes. The attribute *BodyType* (B) has three values: *minivan* (v), *sedan* (s), and *sport* (r). The *Make* (M) can be either *Honda* (h) or *Ford* (f). The *Price* (P) can be *high* (g), *low* (l), or *medium* (d). Finally, *Transmission* (T) can be *automatic* (a) or *manual* (m). An agent could specify her preferences over cars as a PLP-tree T in Figure 1. The tree tells us that *BodyType* is the most important attribute to the agent and that she prefers minivans, followed by sedans and by sport cars. Her next most important attribute is contingent upon what type of cars the agent is considering. For minivans, her most important attribute is *Make*, where she likes Honda more than Ford. Among sedans, her most important attribute is *Price*, where she prefers medium-priced cars over low-priced ones, and those over high-priced ones. Among sport cars, her top priority is *transmission* and she prefers manual to automatic.

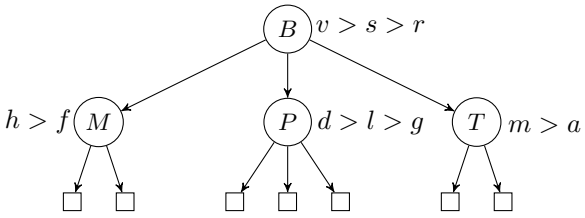
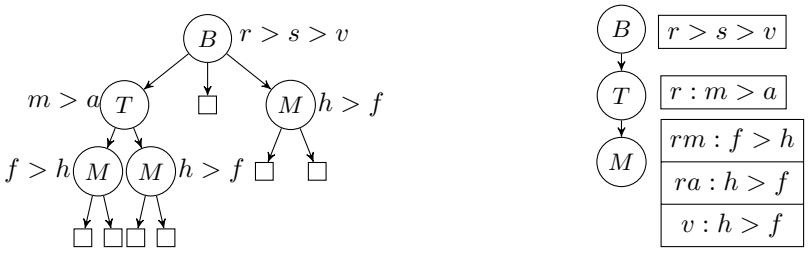


Figure 1: A PLP-tree T over the car domain

Let us consider a Ford sedan with a middle-range price and an automatic transmission ($\langle s, f, d, a\rangle$, in our notation) and a Honda sedan with a high-range price and a manual transmission (that is, $\langle s, h, g, m\rangle$). Writing c_1 and c_2 for the two cars, respectively, and traversing the tree T, we see that cars c_1 and c_2 diverge on the node labeled by attribute P, and that $c_1(P) > c_2(P)$. As a result, we obtain that $c_1 \succ_T c_2$ (c_1 is better than c_2).

In the worst case, the size of a PLP-tree is exponential in the number of attributes in $\mathcal{A}$.

However, a special structure in some PLP-trees allows us to "collapse" them and obtain more compact representations. Let $R \subseteq \mathcal{A}$ be the set of attributes 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, attributes that label nodes on that path appear in the same order in which they appear in $\hat{R}$.



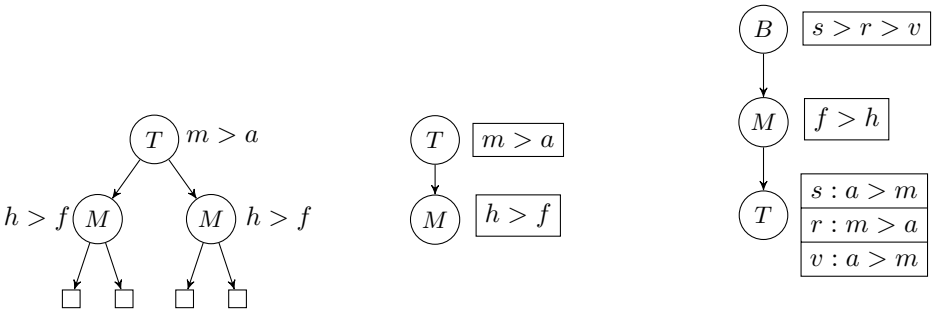
(a) Collapsible PLP-tree (b) UICP PLP-tree

Figure 2: PLP-trees over the car domain

If a PLP-tree T is collapsible, we can represent T by a single path of nodes labeled with attributes according to the order in which they occur in $\hat{R}$, where a node labeled with an attribute X_i is also assigned a *conditional preference table* (CPT) that specifies preferences on X_i, conditioned on values of ancestor attributes in the path. These tables make up for the lost structure of T as different ways in which ancestor attributes evaluate correspond to different locations in the original tree T. Missing entries in the CPT of X_i imply that under the corresponding conditions, all values of X_i are equivalent (equally good). The PLP-tree in Figure 2a is collapsible, and can be represented compactly as a single-path tree with nodes labeled by attributes and CPTs (cf. Figure 2b). The CPT for T (M) indicates that preferences on the values of T (M) depend on B (B and T, respectively), and both of these CPTs are incomplete with missing entries. For instance, the agent is indifferent between automatic and manual transmissions when the body type is minivan and sedan. (Incidentally, the PLP-tree in Figure 1 is collapsible, too.)

Collapsible PLP-trees represented by a single path of nodes are called *unconditional importance* trees or UI trees, for short. The name reflects the fact that the order of attributes on outcome paths is not conditioned on the values of ancestor attributes. A collapsed UI tree may have size substantially smaller than the "uncollapsed" one, especially if local preferences depend on few ancestor attributes. Of particular interest are collapsible trees whose all nodes labeled with the same attribute are assigned the same local preference order. In their collapsed representation, CPTs assigned to nodes consist of a single unconditioned entry. Such UI trees are called UIUP trees, with UP indicating *unconditional preference*. Clearly, the collapsed representation of a UIUP tree is exponentially smaller than its explicit representation. The UIUP tree in Figure 3b is the collapsed representation of the collapsible PLP-tree in Figure 3a.

To stress that local preferences may be conditional, we will use the term UICP trees when referring to UI trees that are not UIUP trees. Among UICP trees, of practical interest are those where the number of parents is bounded by some fixed small integer k independent of p, and the CPTs are complete. We call this type of trees UICP$_k$ PLP-trees. In this case, the sizes of the CPTs and, consequently, the sizes of the trees are polynomial in the number of attributes. An example UICP$_1$ tree is shown in Figure 3c. One can show that our earlier definition of the preference relation $\succeq_T$ given for the "full" representation of PLP-trees can be adapted to work with collapsed PLP-trees (i.e., UI trees).



(a) Collapsible PLP-tree (b) UIUP PLP-tree (c) UICP$_1$ PLP-tree

Figure 3: More PLP-trees over the car domain

In general, PLP trees do not need to be collapsible and the order of attributes on paths from the root to leaves may differ from path to path and so, the importance of attributes is *conditional*. We refer to such PLP trees as conditional importance trees or, CI trees. Depending on whether preferences in CI trees are unconditional or not, we refer to them as conditional importance and unconditional preference trees (or CIUP trees), and conditional importance and conditional preference trees (or CICP trees). We refer to the paper by Liu and Truszczynski [12] for formal definitions and examples of CI PLP-trees.

We now formally define the learning problem we study and comment on its complexity. In the problem we are given a set of examples, that is, expressions $(\alpha, \beta, \succ)$ and $(\alpha, \beta, \approx)$, where α and β are outcomes. Examples of the first type are *strict* examples and of the second type *equivalence* examples. A PLP-tree T satisfies a strict example $(\alpha, \beta, \succ)$ if $\alpha \succ_T \beta$. Similarly, T satisfies an equivalence example $(\alpha, \beta, \approx)$ if $\alpha \approx_T \beta$. The goal is to compute a PLP-tree (of a specified type) that satisfies the maximum number of examples from the input set. We refer to this problem as MaxLearn.

The MaxLearn problem is NP-hard for each of the four classes of PLP-trees we discussed (when applicable, assuming that we learn collapsed representations). This is an easy consequence of the fact that the corresponding decision versions of the problem (asking for the existence of a PLP-tree of a given type satisfying at least k examples from the input set, where k is another input parameter) are NP-complete [12].

2.1 Algorithms learning PLP-trees

We will now outline best-agreement (exact) and greedy (approximation) learning algorithms for the MaxLearn problem. For simplicity, we restrict attention to the case when all examples are strict.

To find the best-agreement model, that is, to compute a PLP-tree (of a specified type) that maximizes the number of satisfied examples, we used answer-set programming (ASP) [14, 15] and its *gringo/clasp* grounder-solver tool [8]. This approach consists of two logical programming modules: the data module describing the dataset (i.e., attributes, domains, outcomes and examples), and the rule module applying an optimization statement to search for a PLP-tree that correctly decides as many examples as possible. Given an instance of the MaxLearn problem expressed as the two modules, the ASP tool *gringo/clasp* computes an answer set encoding the PLP-tree that is a solution to the input instance.

Our method to solve the MaxLearn problem approximately, that is, to compute a PLP-tree that satisfies possibly many examples (but perhaps not the maximum number) is based on

a greedy approach. It is similar to the greedy method proposed by Schmitt and Martignon [16] to learn the so called UIFP trees.[1] A formal description of the greedy algorithm is included in the appendix.

The algorithm has two versions, one to learn UI trees and the other one to learn CI trees. Both versions start with a "configuration" $(\mathcal{E}^{\succ}, \mathcal{A}, n)$, where $\mathcal{E}^{\succ}$ is the input set of all strict examples, $\mathcal{A}$ is the set of attributes of the domain of the problem, and n is a node (initially, a node to serve as the root of the tree to learn). Both versions set $T = n$ and store $(\mathcal{E}^{\succ}, \mathcal{A}, n)$ in an auxiliary set C. Upon termination, T is the root of the learned tree.

UI tree learning: In this case, C consists always of exactly one configuration $(\mathcal{E}^{\succ}, \mathcal{A}, n)$. In each iteration, n is labeled with an attribute $X_l \in \mathcal{A}$ and with a CPT for X_l to maximize the number of examples in $\mathcal{E}^{\succ}$ that are *correctly* decided by X_l under the selected CPT.[2] The algorithm updates $\mathcal{E}^{\succ}$ by removing all examples decided (correctly or incorrectly) by X_l and its CPT, updates $\mathcal{A}$ by removing X_l, creates a new node n', makes n its parent, and updates n to n'. It then processes this new configuration. The process terminates when $\mathcal{E}^{\succ}$ is empty.

We implemented UI tree learning algorithm in two versions in which each CPT consists of a single unconditional preference preference order (to learn UIUP trees), or is a full CPT conditioned on values of at most one attribute (to learn UICP$_1$ trees).

CI tree learning: In each iteration, the algorithm removes one configuration $(\mathcal{E}^{\succ}, \mathcal{A}, n)$ from C. The algorithm picks an attribute X_l and a preference order for X_l at n (not a CPT, as here the structure of the tree is used to model conditional preferences) so that to maximize the number of examples in $\mathcal{E}^{\succ}$ *correctly* decided at this node. If we are learning CIUP trees, the choice of the preference order is restricted: if X_l has already been used as a label before, we have to use the preference order we originally assigned to X_l with all nodes we label with X_l. The algorithm updates $\mathcal{E}^{\succ}$ by removing all examples decided at n (correctly or incorrectly), and $\mathcal{A}$ by removing X_l. At this point, both outcomes of every example in $\mathcal{E}^{\succ}$ have the same value on X_l (otherwise, the example would be decided, correctly or not, by X_l). For each value $x_{l,i}$ of X_l, the algorithm creates a node n_i, makes n the parent of n_i, and sets $\mathcal{E}_i^{\succ}$ to consist of all examples in $\mathcal{E}^{\succ}$ whose both outcomes have value $x_{l,i}$ on X_l. It then adds $(\mathcal{E}_i^{\succ}, \mathcal{A}, n_i)$ to C. When the new set C is computed, the algorithm starts the next iteration. Configurations with the empty set of examples become leaves and the process ends when they are no more configurations to process.

We note that CIUP tree learning algorithm returns different trees depending on the order in which configurations in C are considered. We studied two versions based on the depth-first order (using a stack to represent C), and the breadth-first order (using a queue to represent C), resulting in CIUP$_d$ and CIUP$_b$ trees, respectively.

2.2 Experiments

We studied the performance of our algorithms by experimenting with them on datasets we derived from publicly available *classification* datasets in the University of California at Irvine Machine Learning Repository. The datasets are listed in Table 1 and their characteristics (the numbers of attributes and outcomes, and strict and equivalence examples) are given in Table 2. The appendix describes the datasets in more detail.

First, we discuss learning UIUP PLP-trees using the best-agreement and greedy methods. The goal is to compare the accuracy of both methods. This is important as the best-agreement

[1] They are UIUP trees in which the order on the values of the domain of every attribute is fixed *a priori* and must be used in the tree.

[2] A strict example is decided at a node, if the two outcomes of the example have different value on the attribute labeling the node. It is correctly decided, if the (strict) relation in the example agrees with the local preference relation on the values of the outcomes in the example.

Table 1: Classification datasets in UCI Machine Learning Repository used to generate preference datasets

Preference Datasets	Classification Datasets in UCI MLR
BCW	Breast Cancer Wisconsin
CE	Car Evaluation
CA	Credit Approval
GC	Statlog (German Credit Data)
IN	Ionosphere
MM	Mammographic Mass
MS	Mushroom
NS	Nursery
SH	SPECT Heart
TTT	Tic-Tac-Toe Endgame
VH	Statlog (Vehicle Silhouettes)
WN	Wine

Table 2: Description of preference datasets in the library

Dataset	p	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{E}^{\succ}\rvert$	$\lvert\mathcal{E}^{\approx}\rvert$
BCW	9	270	9,009	27,306
CE	6	1,728	682,721	809,407
CA	10	520	66,079	68,861
GC	10	914	172,368	244,873
IN	10	118	3,472	3,431
MM	5	62	792	1,099
MS	10	184	8,448	8,388
NS	8	1,266	548,064	252,681
SH	10	115	3,196	3,359
TTT	9	958	207,832	250,571
VH	10	455	76,713	26,572
WN	10	177	10,322	5,254

method, because of its complexity, can only be used on relatively small example sets.

For a dataset D (where D is one of the twelve datasets we studied), we fix the size of the training set to t, where $1 \leq t \leq 250$. Then, we randomly pick $TR_D \subseteq \mathcal{E}^{\succ}$, where $|TR_D| = t$, as the set of *training* examples, and use $TE_D = \mathcal{E}^{\succ} \setminus TR_D$ as the set of *testing* examples. Based on training examples in TR_D, we learn a UIUP PLP-tree T_{BA} using the best-agreement method, and a UIUP PLP-tree T_G using the greedy heuristics. We then verify the models T_{BA} and T_G on testing examples in TE_D and compute the *accuracy* of each method, as the percentage of strict examples in TE_D decided correctly by the corresponding tree. For each t, $1 \leq t \leq 250$, we repeat this process 20 times and compute the average accuracies. Figure 4 shows the *learning curves* (the accuracies as the function of the size of the training set) for the best-agreement method (BA-UIUP) and the greedy algorithm (G-UIUP) for the datasets CE, IN, MS and WN. We show the accuracies for the two methods on all datasets when $t = |TR_D| = 250$ in Table 3.

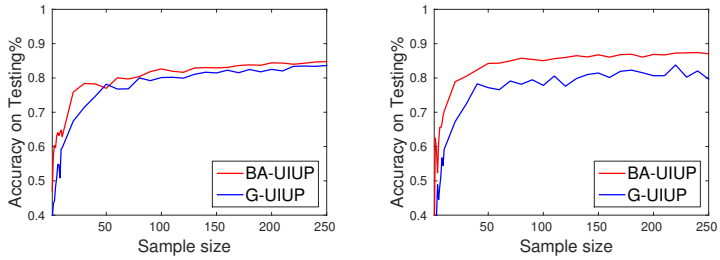
This experiment shows that, when the number of training examples is small, the greedy approach achieves accuracy comparable with that of the best-agreement method. The results summarized in Table 3 show that (1) the greedy algorithm already achieves accuracy exceeding 85% on six datasets (most notably, the accuracy of 95.5% on WN); and (2) the greedy algorithm performs very close to the best-agreement method, with the difference within 2 percentage points on *all but two datasets*, IN and MS. Examining the learning curves in Figure 4(a) and (d) (they are representative of 10 out of 12 datasets — all but IN and MS), we observe that the greedy algorithm works well compared with the best-agreement method across the range of the training set sizes. The learning curves for the two datasets on which the greedy method lags behind the best-agreement one are shown in Figure 4(b) and (c).

Since the best-agreement method quickly fails as the training sample size grows (the problem is NP-hard), in experiments with large learning sets we only used the greedy heuristics. As demonstrated above, the greedy heuristics is a good alternative to the best-agreement method. For a dataset D, we generate $TR_D \subseteq \mathcal{E}^{\succ}$ as the training set, and use $TE_D = \mathcal{E}^{\succ} \setminus TR_D$ as the testing set. We learn UIUP, UICP$_1$, CIUP$_b$, and CIUP$_d$ trees based on TR_D using the greedy heuristics, and then we verify the trees on the testing set TE_D, computing their accuracy. Table 4 shows the accuracies of trees learned from training sets of size 70% of the of $\mathcal{E}^{\succ}$.[3]

[3] As in the previous experiment, we computed the learning curves by varying the size of the training set up to 70% of the size of $\mathcal{E}^{\succ}$. The curves show similar behavior to those presented earlier — the accuracy increases with the size of the training set, but gets close to the maximum accuracy already for relatively small training sets.

Table 3: Accuracy (percentage of correctly handled testing examples) for UIUP PLP-trees learned using the best-agreement and the greedy methods on the learning data (250 of $\mathcal{E}^{\succ}$)

Dataset	BA-UIUP	G-UIUP
BCW	88.4	88.2
CE	84.8	83.6
CA	91.1	89.3
GC	72.2	72.2
IN	87.0	79.6
MM	87.5	86.8
MS	84.8	70.3
NS	91.8	91.7
SH	93.2	92.6
TTT	72.1	71.9
VH	76.8	76.6
WN	96.0	95.5



(a) CE (b) IN

(c) MS (d) WN

Figure 4: Learning UIUP PLP-trees

From Table 4 we note that for the greedy algorithm: (1) for all datasets, there is a clear gain in the accuracies for the UIUP trees learned from larger training sets (cf. Table 3); (2) for all but one dataset (IN), the UICP_1 trees, which allow for simple conditional preference statements, are more accurate than the UIUP trees; (3) both the CIUP_b and CIUP_d models are more accurate than the UIUP models for all but one dataset (MM); and (4) the most general class CICP achieves the best accuracies among all four classes of PLP-trees across all datasets.

The size of a PLP-tree is measured by the total number of preferences in the CPTs in the tree. Clearly, for UIUP, CIUP and CICP trees, it is also the number of non-leaf nodes in the tree. For UICP trees it is the total number of rows in all conditional preference tables in the tree. It is desirable to learn trees that are accurate but small. Trees of a small size provide qualitative insights into the structure and properties of the preference order of a user.

Table 4: Accuracy percentages on the testing data (30% of $\mathcal{E}^{\succ}$) for all four classes of PLP-trees, using models learned by the greedy algorithm from the learning data (the other 70% of $\mathcal{E}^{\succ}$)

Dataset	UIUP	UICP_1	CIUP_b	CIUP_d	CICP
BCW	90.7	91.4	91.0	90.7	91.4
CE	85.8	86.0	85.8	85.9	86.0
CA	91.4	91.7	91.6	92.0	92.2
GC	74.3	74.6	74.3	74.5	75.7
IN	87.1	86.9	87.2	88.5	90.4
MM	88.2	89.5	87.3	86.9	90.0
MS	71.6	74.2	77.1	75.6	76.6
NS	92.9	93.0	93.0	93.0	93.0
SH	93.4	94.9	95.4	94.8	95.7
TTT	73.9	74.5	74.4	75.4	76.2
VH	79.2	80.4	80.3	80.0	81.2
WN	95.5	97.8	97.8	97.5	97.8

Table 5: Maximum sizes of trees for all the classes and the training sample sizes for all datasets

Dataset	UIUP	UICP_1	CI	$\lvert\mathcal{E}^{\succ}_{train}\rvert$
BCW	9	33	87,381	6,306
CE	6	21	853	477,904
CA	10	37	91,477	46,255
GC	10	37	349,525	120,657
IN	10	19	1,023	2,430
MM	5	17	341	554
MS	10	37	91,477	5,913
NS	8	29	7,765	383,644
SH	10	19	1,023	2,237
TTT	9	25	9,841	145,482
VH	10	37	349,525	53,699
WN	10	37	349,525	7,225

Table 6: Average sizes of trees learned by the greedy algorithm from the training data (70% of $\mathcal{E}^{\succ}$)

Dataset	UIUP	UICP_1	CIUP_b	CIUP_d	CICP
BCW	6.7	21.8	19.8	28.0	25.7
CE	6.0	17.0	73.2	108.9	109.5
CA	9.0	24.7	31.3	78.6	81.1
GC	9.7	36.0	49.8	210.3	190.0
IN	9.6	17.2	19.8	31.5	30.6
MM	4.5	14.7	8.3	10.8	10.0
MS	7.6	20.7	15.7	22.7	16.3
NS	8.0	25.7	56.2	121.0	116.9
SH	8.4	13.7	13.0	18.4	19.0
TTT	8.0	21.8	36.8	126.8	115.2
VH	9.0	32.7	33.9	101.3	105.4
WN	5.1	13.3	14.2	16.9	14.6

The size of a PLP-tree learned by the greedy algorithm is bounded by the number of training examples. On the other hand, it never exceeds the size of the largest possible tree for a domain it models. These maxima are shown for each dataset in Table 5. The maximum for CI trees is the common maximum for CIUP_b, CIUP_d and CICP trees. The last column in the table shows the size of the training example set used (70% of all examples).

Table 6 shows average size of trees learned by our greedy algorithm (for each dataset and for each class of trees considered). The results indicate that the learned trees have indeed relatively small sizes when compared to the upper bounds implied by Table 5. The difference is drastic for CIUP_b, CIUP_d and CICP trees, where trees we learn have sizes that are small fractions of the maximum possible size they potentially might have. For UIUP trees and UICP_1 trees, the difference is smaller (these trees, because of their structure, are very small to start with), yet even there is some cases the learned trees have sizes below 80% of the maximum size and occasionally are much smaller (for instance for the WN dataset). These small-size trees can provide explicit insights into the importance the user assigns to attributes when deciding between outcomes, and into how her preferences of attributes depend on preferences on the more important ones.

We also observe that the sizes of learned CIUP_b trees are always smaller than the sizes of the learned CI trees of the other two types. In some cases (datasets GC, NS, TTT, VH), they are significantly smaller. Given that the accuracies of learned CIUP_b and CIUP_d trees are very

close to each other, and the accuracies of the learned CIUP$_b$ and CICP trees differ by more than 2 percentage points in only one case (GC), the results suggests that CIUP$_b$ trees provide a particularly attractive preference model. The results are well aligned with the intuition that when using CIUP trees, agents build them level by level in a breadth-first fashion.

Another important property of greedy algorithms is that they work fast even on large training sets. Our timing experiments show that they scale up linearly with the size of the training set. In contrast, the best agreement method scales up exponentially and times out (with the timeout set to 120 seconds) even on small training sets with 250 elements.

Closing this section, we provide a brief comparison between PLP-trees and decision trees, a commonly-used classification model in machine learning. Decision trees can be used as classifiers that, given two outcomes, can tell if an outcome is better or worse than another. Our experimental results show that decision trees are generally better on predicting preferences between outcomes than PLP-trees are, although the difference rarely exceeds 3 percentage points on our data sets. However, PLP-trees offer not only a quick way to determine dominance (the order between two outcomes) but also insights into the structure of the reasoning process of the decision maker. They point to importance of attributes and conditional dependencies between them, and explicitly identify optimal outcomes. This information is hard to glean out of the decision-tree model for the dominance relation.

3 Partial Lexicographic Preference Forests

As we see from Table 4, our approximation method achieves high accuracy (above 85%) on most of the datasets for all four types of PLP-trees. However, on some datasets such as MS, PLP-trees that we learn have accuracy below 80% across all classes of trees. In an effort to improve on this, we introduce the notion of a *PLP-forest*, that is, a *collection* of PLP-trees. Let $F = \{T_1, \ldots, T_n\}$ be a PLP-forest. We say that F is a $\mathcal{C}$ PLP-forest, where $\mathcal{C}$ is one of the four classes UIUP, UICP$_1$, CIUP and CICP, if F consists exclusively of $\mathcal{C}$ PLP-trees.

3.1 Aggregating PLP-Trees in a PLP-Forest

We use the *pairwise majority rule* (PMR) to aggregate orders defined by trees in a forest. The choice of PMR as the aggregation rule is motivated by three considerations. First, plurality was used in the related work on random forest learning [4] that motivated and influenced our ideas behind PLP forests and PLP forest learning. Second, the task we have at hand is to determine the preferences between outcomes, so PMR is well aligned with this task (the outcome that "wins" on more orders "wins" overall). Finally, the PMR is intuitive and easy to implement.

Let us denote by $N_F(o_1, o_2) = |\{T \in F : o_1 \succ_T o_2\}|$ the number of trees in the forests where the outcome o_1 is preferred to the outcome o_2. Given a forest F, and two outcomes o_1 and o_2, we say that $o_1 \succ_F^{PMR} o_2$ iff $N_F(o_1, o_2) > N_F(o_2, o_1)$, and that $o_1 \approx_F^{PMR} o_2$ iff $N_F(o_1, o_2) = N_F(o_2, o_1)$.

In some cases, PMR may lead to the so-called *Condorcet's Paradox*, where the strict $\succ_F^{PMR}$ relation contains a cycle. It is not the case for our datasets, which we created in such a way that the Condorcet's Paradox is prevented from happening.

3.2 Experimentation

First, we show results for UIUP PLP-forests using the best-agreement learning and the greedy heuristics. In each experiment, we randomly partitioned a dataset into training set (70%) and testing set (30%). We used the training set to learn a forest of 5000 trees, where each tree is learned from 50 examples selected with replacement and uniformly at random from the training set. We repeated that process 20 times (learned 20 forests), and computed their

Table 7: Accuracy percentages on the testing data (30% of $\mathcal{E}^{\succ}$) for UIUP trees and forests of 5000 UIUP trees, using the greedy and the best-agreement algorithms from the learning data (the other 70% of $\mathcal{E}^{\succ}$)

Dataset	G+Tree	G+Forest	BA+Forest
BCW	90.7	93.4	95.1
CE	85.8	91.9	89.2
CA	91.4	91.5	93.1
GC	74.3	75.4	77.9
IN	87.1	83.0	92.5
MM	88.2	89.1	90.8
MS	71.6	78.8	90.2
NS	92.9	93.2	94.0
SH	93.4	93.7	94.9
TTT	73.9	75.1	77.2
VH	79.2	82.7	81.9
WN	95.5	95.8	96.9

Table 8: Accuracy percentages on the testing data (30% of $\mathcal{E}^{\succ}$) for all four classes of PLP-forests of 5000 trees, using the greedy algorithm from the learning data (the other 70% of $\mathcal{E}^{\succ}$)

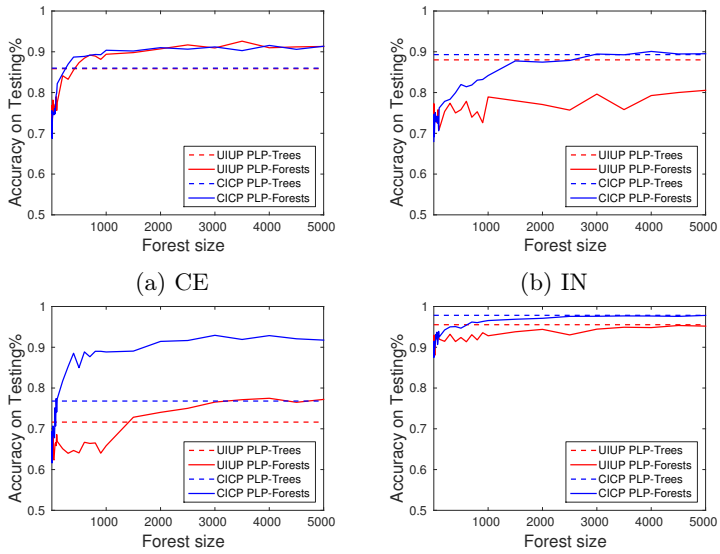
Dataset	UIUP	UICP_1	CIUP_b	CIUP_d	CICP
BCW	93.4	94.1	93.7	94.1	94.0
CE	91.9	88.3	91.4	89.7	91.4
CA	91.5	91.6	92.8	92.9	93.0
GC	75.4	73.8	76.1	76.1	76.2
IN	83.0	87.9	89.3	89.4	89.5
MM	89.1	90.1	90.0	90.1	90.2
MS	78.8	87.2	92.2	92.2	91.8
NS	93.2	89.9	93.3	93.4	93.4
SH	93.7	93.5	93.6	93.6	93.7
TTT	75.1	75.2	76.6	76.5	76.9
VH	82.7	81.8	83.2	83.2	83.4
WN	95.8	95.4	97.5	97.8	97.8

accuracies (using the testing set), as well as the average accuracies over all 20 forests learned. The averages are given in Table 7 (we write BA and G to indicate the methods used).

We see that G+Forest outperforms G+Tree on all but one dataset (i.e., IN). This indicates the gain of using a forest of diverse trees against a single tree for UIUP. Similarly, we observe that BA+Forest outperforms G+Forest on all datasets but one (CE). This points to another advantage of PLP forest learning: they achieve good accuracy even when individual trees are learned from small example sets and so, the best-agreement learning becomes practical.

Second, we show results of PLP-forest learning with the greedy heuristics and the five types of PLP-forests (under the same setting as before). The results are shown in Table 8. Comparing with Table 4, we see that UICP_1 trees do not lend themselves well to the use in forests, the accuracies for individual UICP_1 trees are higher than for forests of UICP_1 trees for five out of 12 datasets. However, for all other types of trees, the idea of learning forests of such trees is very effective. We get improvements in the accuracy on all datasets but one for UIUP and CIUP_d trees, and in all but two datasets for CIUP_b and CICP trees. In the case of the dataset MS, the improvements provided by forest learning are particularly significant.

We also studied how the accuracy of PLP-forests changes with the number of their PLP-trees. In Figure 5, we show the results for UIUP and CICP PLP-forests for selected datasets.



(a) CE (b) IN

(c) MS (d) WN

Figure 5: Learning PLP-forests

Examining Figure 5, we note that with even smaller forests, consisting of 2000 forests, the accuracies are already very close to those we observe for forests consisting of 5000 trees. That suggests that much larger forests would not offer any additional boost in the accuracy. The figure also shows that the number of trees needed in a forest in order to offer a better accuracy than that of an individual tree varies (for only one case with dataset IN and class UIUP, we do not see forests of trees surpass individual trees in accuracy).

4 Conclusion and Future Work

We considered learning *partial lexicographic preference trees*, or *PLP-trees*. We showed that PLP-trees are expressive preference models that can be used to accurately model preferences arising in practical situations, and that high-accuracy PLP-trees can be effectively computed. We also proposed and studied a variant of the model based on the concept of a *PLP-forest*, a *collection* of PLP-trees, where the preference order specified by a PLP-forest is obtained by aggregating the orders of its PLP-trees. We proposed and implemented the best-agreement and greedy algorithms to learn PLP-trees and PLP-forests. To support experimentation, we used datasets that we adapted to the preference learning setting from existing classification datasets.

Our results demonstrated the potential of both approaches. For learning single trees, our results show the effectiveness of the greedy heuristics, and identify CIUP_b trees as offering both high accuracy and small tree sizes. Learning PLP-forests improves accuracy and yields useful models even when individual trees are learned from small example sets. That allows us to use the best-agreement method for learning PLP forests, the method inapplicable when example sets are large.

In the future, we will study *partially* collapsible PLP-trees where some subtrees are collapsible while the whole tree is not. We will explore expanding our preference learning library with real-world datasets obtained by experiments involving human subjects. Finally, we plan to implement and experiment with other aggregators for PLP-forests, and compare with our

results using the desirable and intuitive majority rule.

Acknowledgments

This work was partially supported by the NSF grant IIS-1618783.

References

[1] Richard Booth, Yann Chevaleyre, Jérôme Lang, Jérôme Mengin, and Chattrakul Sombattheera. Learning conditionally lexicographic preference relations. In *ECAI*, pages 269–274, 2010.

[2] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. Cp-nets: a tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21(1):135–191, 2004.

[3] Michael Bräuning and Eyke Hüllermeier. Learning conditional lexicographic preference trees. *Preference learning: problems and applications in AI*, page 11, 2012.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Yannis Dimopoulos, Loizos Michael, and Fani Athienitou. Ceteris paribus preference elicitation with predictive guarantees. In *IJCAI*, volume 9, pages 1–6. Citeseer, 2009.

[6] Niall M Fraser. Ordinal preference representations. *Theory and Decision*, 36(1):45–67, 1994.

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

[8] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.

[9] Joshua T Guerin, Thomas E Allen, and Judy Goldsmith. Learning cp-net preferences online from user queries. In *Algorithmic Decision Theory*, pages 208–220. Springer, 2013.

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

[11] Jérôme Lang and Jérôme Mengin. The complexity of learning separable ceteris paribus preferences. In *IJCAI*, pages 848–853, 2009.

[12] Xudong Liu and Miroslaw Truszczynski. Learning partial lexicographic preference trees over combinatorial domains. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1539–1545. AAAI Press, 2015.

[13] Xudong Liu and Miroslaw Truszczynski. Reasoning with preference trees over combinatorial domains. In *Algorithmic Decision Theory*, pages 19–34. Springer, 2015.

[14] Victor W Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, pages 375–398. Springer, 1999.

[15] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[16] Michael Schmitt and Laura Martignon. On the complexity of learning lexicographic strategies. *The Journal of Machine Learning Research*, 7:55–83, 2006.

[17] Nic Wilson. Efficient inference for expressive comparative preference languages. In *IJCAI 2009, Proceedings of the International Joint Conference on Artificial Intelligence, Pasadena, California, Usa, July*, pages 961–966, 2009.

Appendix

Greedy learning algorithm

Algorithm 1: The *greedy* algorithm that learns a PLP-tree

Input: C: a configuration of items $(\mathcal{E}^{\succ}, \mathcal{A}, n, \Delta)$, where $\mathcal{E}^{\succ}$ is the set of strict example to be decided, $\mathcal{A}$ the set of available attributes, n an unlabeled node to consider next, and Δ a Boolean value indicating the type of PLP-trees (UI or CI) to be learned, and $T = n$: an unlabeled node for which a PLP-tree is to be learned.

Output: A PLP-tree T over $\mathcal{A}$.

$(\mathcal{E}^{\succ}, \mathcal{A}, n, \Delta) \leftarrow$ Pop an item from C;
if $\mathcal{E}^{\succ} = \emptyset$ **then**
  Label n as a leaf;
  if C *is empty* **then**
    return;
  end
else
  $(X_l, CPT(X_l)) \leftarrow$ Pick $X_l \in \mathcal{A}$ and $CPT(X_l)$ that correctly decides the maximum number of examples in $\mathcal{E}^{\succ}$;
  Label n with tuple $(X_l, CPT(X_l))$;
  $\mathcal{E}^{\succ} \leftarrow \mathcal{E}^{\succ} \backslash \{e \in \mathcal{E}^{\succ} : \alpha_e(X_l) \neq \beta_e(X_l)\}$;
  $\mathcal{A} \leftarrow \mathcal{A} \backslash \{X_l\}$;
  if $\Delta =$ ***true*** **then**
    Create an edge u and an unlabeled node n' such that $u = \langle n, n' \rangle$;
    Push item $(\mathcal{E}^{\succ}, \mathcal{A}, n', \Delta)$ onto C;
  else
    for $i \leftarrow 1$ *to* $|D_l|$ **do**
      Create an edge u_i and an unlabeled node n_i such that $u_i = \langle n, n_i \rangle$;
      $\mathcal{E}_i^{\succ} \leftarrow \{e \in \mathcal{E}^{\succ} : \alpha_e(X_l) = \beta_e(X_l) = x_{l,i}\}$;
      Push item $(\mathcal{E}_i^{\succ}, \mathcal{A}, n_i, \Delta)$ onto C;
    end
  end
end
greedy(C, T);

Preference Library — Datasets Used in Experimentation

We now describe the datasets we used in our study of learning algorithms we present later. These datasets were generated from publicly available classification datasets developed by the machine learning community. When constructing the datasets, we limited the number of attributes in outcomes to ten and the sizes of attribute domains to four.

Classification datasets associate with each outcome α a label $l(\alpha)$. If there is a total (pre)order relation on the labels, say $\succeq$, we can use this relation to produce preference examples out of classification examples. Namely, for each pair of outcomes α and β from the classification dataset, if $l(\alpha) \succ l(\beta)$, we take $(\alpha, \beta, \succ)$ as a strict example, and if $l(\alpha) = l(\beta)$, we take $(\alpha, \beta, \approx)$ as an equivalence example.[4] Throughout the paper, we write p for the number

[4]Clearly, our preference datasets do not contain incomparability examples. This is not a limitation in our work as the preference models we learn represent total preorders.

of attributes in a dataset, $\mathcal{X}$ for the set of outcomes, $\mathcal{E}$ for the set of examples, and $\mathcal{E}^{\succ}$ and $\mathcal{E}^{\approx}$ for the sets of strict and equivalence examples, respectively.

At present, our preference library consists of twelve datasets obtained from the classification datasets listed in Table 1. In ten of them there is a natural order on the labels. For the other two of them namely, VH and WN, there is no domain-specific natural order on the labels. In these two cases, to generate examples we fixed a preference order on the labels arbitrarily (see below). We discuss three preference datasets (CE, VH and WN) in detail and provide a summary description of the remaining ones in Table 2, where we use $|\cdot|$ to denote the size of a set.

BCW The BCW dataset has 270 outcomes over 9 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “benign" are better than those by “malignant." For equivalent examples, we have that outcomes labeled by “benign" are equivalent to one another, so are those labeled by “malignant."

CE he CE dataset has 1728 outcomes over 6 attributes. o generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “vgood" are better than those by “good," which are better than those by “acc," which are preferred to those by “unacc."

CA The CA dataset has 520 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “+" (positive) are better than those by “-" (negative).

GC The GC dataset has 914 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “1" (good) are better than those by “2" (bad).

IN The IN dataset has 118 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, those by “b" (bad).

MM The MM dataset has 118 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “0" (benighn) are better than those by “1" (malignant).

MS The MS dataset has 184 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “e" (edible) are better than those by “p" (poisonous).

NS The NS dataset has 184 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “spec_prior" are better than those by “priority," which are better than those by “very_recom," which are preferred to those by “recommend," which again are better than those by “not_recom."

SH The SH dataset has 115 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “0" (positive) are better than those by “1" (negative).

TTT The TTT dataset has 958 outcomes over 9 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “positive" are better than those by “negative".

VH The VH dataset has 455 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “bus" are better than those by “opel," which are better than those by “saab," which are preferred to those by “van."

WN The WN dataset has 177 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by “1" are better than those by “2," which are better than those by “3."