Learning Lexicographic Preference Trees from Positive Examples

Hélène Fargier, Pierre-François Gimenez, Jérôme Mengin
IRIT, CNRS, University of Toulouse, 31000 Toulouse, France
{fargier, pgimenez, mengin}@irit.fr

Abstract

This paper considers the task of learning the preferences of users on a combinatorial set of alternatives, as it can be the case for example with online configurators. In many settings, what is available to the learner is a set of positive examples of alternatives that have been selected during past interactions. We propose to learn a model of the users' preferences that ranks previously chosen alternatives as high as possible. In this paper, we study the particular task of learning conditional lexicographic preferences. We present an algorithm to learn several classes of lexicographic preference trees, prove convergence properties of the algorithm, and experiment on both synthetic data and on a real-world bench in the domain of recommendation in interactive configuration.

1 Introduction

Modern, interactive decision support systems like recommender systems or configurators often handle a very large set of possible decisions/alternatives. The task of finding the alternatives that best suit their preferences can be challenging for users, but the system can guide them towards their optimal decision if it has some knowledge of their preferences. In many settings, the users' preferences are not known in advance. This is especially the case of systems that enable anonymous users to browse the catalogues: such systems must be able to acquire users' preferences.

That is why preference learning has emerged in the last decade as an important field; many interesting results are reported in e.g. the book edited by (Fürnkranz and Hüllermeier 2011), or the proceedings of recent Preference Learning or DA2PL (Decision Aid to Preference Learning) workshops. A general problem is: given a set of observed preferences, induce a model of preferences that best explains these observations, within a certain class of models. As input, it is often assumed that the observed preferences are given as a set of pairwise comparisons or partial rankings of alternatives (Joachims 2002); or can be elicitated online by asking the user to choose between two alternatives (Viappiani, Faltings, and Pu 2006; Koriche and Zanuttini 2009).

But in some circumstances, such input is not available. This is especially the case of some anonymous on-line configurators, where little information is stored about interactions. However, e-commerce companies in general keep a history of past sales. Sold items have been chosen by users, so they must be ranked high in their preferences, but not necessarily in the very top; indeed a user may be led to eventually choose an item which is not the optimal one in her preference order: for instance because of the difficulty to grasp all possible options, a phenomenon called "mass confusion" (Huffman and Kahn 1998), because of the influence of an advertisement, or because her preferred item is unavailable. Yet, this list of highly ranked items does provide information about the users' preferences.

Our aim in this paper is to study how the users' preferences can be learnt from this sales history. Note that this is different from a classification problem where one would want to separate alternatives between, say, acceptable ones and not acceptable ones. In our problem, we want to rank the alternatives. If, for instance, the colour red appears more often in the sales history than the colour yellow, then we want to induce a model that ranks alternatives with the colour red higher than alternatives with the colour yellow – maybe in association with some other criteria.

Research on the representation and learning of preferences has brought forward several types of models. Numerical models, like linear ranking functions or additive utilities (Joachims 2002; Freund et al. 2003; Schiex, Fargier, and Verfaillie 1995; Gonzales and Perny 2004; Braziunas 2005), are rich families of models, especially if one allows high-dimensional feature spaces. Research in Artificial Intelligence has also brought forward ordinal models, like CP-nets (Boutilier et al. 2004) and several extensions or variants. Lexicographic preferences are another family of ordinal models. This kind of preference is based on the importance of the attributes: when comparing two outcomes, their values for the most important attribute are compared; if the two outcomes have different values for that attribute, then the one with the preferred value is deemed preferable to the other; otherwise one looks at the next most important attribute, and so on. This model can be extended by allowing the preferences on the values of an issue to depend on the values of more important ones. The relative importance of issues is no longer a linear order, but a "lexicographic preference" tree (Fraser 1993; 1994; Brewka and others 2006; Wallace and Wilson 2009; Booth et al. 2010).

Lexicographic preference trees are the model that we

choose in this paper for two main reasons. First, this is an ordinal model, which is sufficient to represent a ranking of alternatives. Furthermore, the restricted expressivity of the lexicographic preference trees makes them easier to learn while being generally an accurate representation of human behaviours (Gigerenzer and Goldstein 1996). Finally, one can quickly (in polytime) perform some interesting requests for recommendation, such as finding an optimal object or an optimal value for some attribute.

Learning lexicographic preference models have been studied by e.g. (Schmitt and Martignon 2006; Dombi, Imreh, and Vincze 2007; Yaman et al. 2008), while (Booth et al. 2010; Bräuning and Hüllermeyer 2012; Bräuning et al. 2017; Liu and Truszczynski 2015) studied learning of lexicographic preference trees. But all these works assume sets of pairwise comparisons as inputs, while we seek to learn from sales history. Preference learning is also different from learning a (lexicographic) rank-based classifier, as proposed by e.g. (Flach and Matsubara 2007): the latter task takes as input a set of positive and negative instances of a concept.

The paper is structured as follows. The next Section summarizes the lexicographic preference tree models and introduces the LP-tree classes we aim to learn. Section 3 details the probabilistic model on which our approach is based. The algorithmics is developed in Section 4. Section 5 describes experiments on synthetic data and Section 6 on a real-world dataset in the domain of recommendation in interactive configuration.

2 Lexicographic preference trees

Notations

We consider a combinatorial domain over a set $\mathcal{X}$ of n discrete attributes, each attribute X having a finite domain $\underline{X}$. An *outcome* is an instantiation of $\mathcal{X}$. We use upper-case, bold letters to represent tuples of attributes; if $\mathbf{U}$ is such a tuple, then the set of assignments of $\mathbf{U}$ is denoted by $\underline{\mathbf{U}}$, and the corresponding lower-case letter will generally denote such an assignment: $\mathbf{u} \in \underline{\mathbf{U}}$. A (complete) outcome is therefore a tuple $\mathbf{o} \in \prod_{X \in \mathcal{X}} \underline{X}$; we denote by $\underline{\mathcal{X}}$ the set of all of them. $\mathbf{o}[\mathbf{U}]$ denotes the projection of $\mathbf{o}$ onto $\mathbf{U}$: it is a partial outcome, i.e. an instantiation of $\mathbf{U}$; we say that $\mathbf{o}$ extends $\mathbf{o}[\mathbf{U}]$.

If $\mathbf{U}$ and $\mathbf{V}$ are disjoint tuples of attributes, $\mathbf{UV}$ will denote their concatenation – with similar notations for assignments: if $\mathbf{u} \in \underline{\mathbf{U}}$ and $\mathbf{v} \in \underline{\mathbf{V}}$ then $\mathbf{uv} \in \underline{\mathbf{UV}}$. If $\mathbf{U}$ and $\mathbf{V}$ are not disjoint, we say that $\mathbf{u}$ and $\mathbf{v}$ are compatible if $\mathbf{u}[\mathbf{U} \cap \mathbf{V}] = \mathbf{v}[\mathbf{U} \cap \mathbf{V}]$.

A preference is modelled as a transitive binary relation over the domain of interest, $\underline{\mathcal{X}}$ in our case. We consider in this paper that indifference is not allowed; the relation is said to be strict and we denote it $\succ$: $\mathbf{o} \succ \mathbf{o}'$ indicates that $\mathbf{o}$ is (strictly) preferred to $\mathbf{o}'$. We denote by $\mathsf{rank}(\succ, \mathbf{o})$ the rank of outcome $\mathbf{o}$ in the relation $\succ$: the "best" outcome has rank 1, the worst has rank $|\underline{\mathcal{X}}|$.

Lexicographic preference trees

In the formal model proposed by (Booth et al. 2010), a *lexicographic preference tree*, or LP-tree for short, is composed of two parts: a rooted tree indicating the relative importance of the attributes, and tables indicating how to compare outcomes that agree on some attributes. Each node of the importance tree is labelled with an attribute $X \in \mathcal{X}$, and is either a leaf of the tree, or has one single, unlabelled outgoing edge, or has $|\underline{X}|$ outgoing edges, each one being labelled with one of these values. No attribute can appear twice in a branch. For a given node N, $\mathsf{Anc}(N)$ denotes the set of attributes that label nodes above N. The values of attributes that are at a node above N with a labelled outgoing edge influence the preference at N. We denote by $\mathsf{Inst}(N)$ the set of nodes above N with a labelled outgoing edge and $\mathsf{inst}(N)$ the tuple of values of the labels between the root and N.

Example 1. An example of a LP-tree is depicted in Figure 1a. Let N be the leftmost leaf labelled C, we have $\mathsf{Anc}(N) = \{A, B\}$ and $\mathsf{inst}(N) = a$.

Moreover, one *conditional preference table* $\mathsf{CPT}(N)$ is associated to each node N of the tree: if X is the attribute that labels N, then the table contains (consistent) rules of the form $\mathbf{v} : >$, where $\mathbf{v}$ is a (possibly partial) instantiation of the attributes in $\mathsf{Anc}(N) \setminus \mathsf{Inst}(N)$, and $>$ is a strict total order $>$ over $\underline{X}$. Every LP-tree $\mathcal{L}$ induces a (possibly partial) strict order over $\underline{\mathcal{X}}$, denoted $\succ_{\mathcal{L}}$, as follows: for any node N labelled by X, consider a pair of complete outcomes $\mathbf{o}$ and $\mathbf{o}'$ such that $\mathbf{o}[\mathsf{Inst}(N)] = \mathbf{o}'[\mathsf{Inst}(N)] = \mathsf{inst}(N)$ and $\mathbf{o}[\mathbf{V}] = \mathbf{o}'[\mathbf{V}] = \mathbf{v}$ for some rule $\mathbf{v} : >$ in $\mathsf{CPT}(N)$ with $\mathbf{v} \in \underline{\mathbf{V}}$; N is said to *decide* the pair $(\mathbf{o}, \mathbf{o}')$, and $\mathbf{o} \succ_{\mathcal{L}} \mathbf{o}'$ if and only if $\mathbf{o}[X] > \mathbf{o}'[X]$.

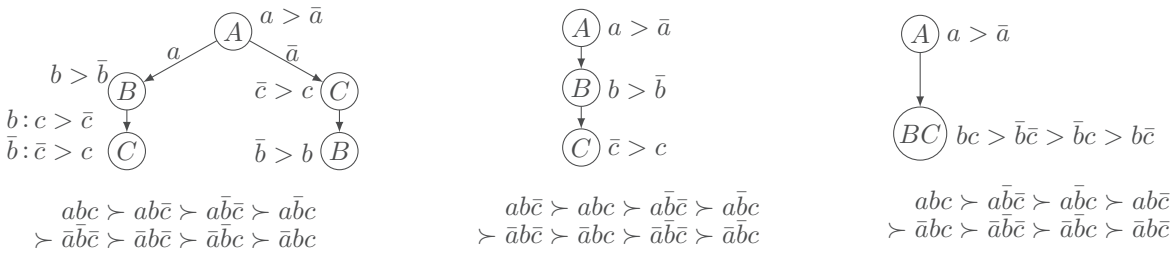
A branch of a tree is *complete* iff all attributes appear in it; if all the branches of a tree are complete, then the tree itself is said to be complete. The preference relation induced by a LP-tree is total if and only if the tree is complete (Bräuning and Hüllermeyer 2012). In this case, every outcome has a well-defined rank in the preference relation (and there is only one optimal outcome, ranked 1). (Lang, Mengin, and Xia 2012) have shown that it can be computed in polytime and that, for a given rank, finding an outcome with that rank can also be done in polytime. In the following, we will deal with complete trees and ranking only.

Linear LP-trees and k-LP-trees

In this paper, in addition to general LP-trees, we will be interested in a syntactic restriction called linear LP-tree and a extension called k-LP-tree.

A *linear LP-tree is a LP-tree with a single branch and unconditional preference rules only*, like the tree in Figure 1b. It is a strong expressive restriction: linear LP-trees represent the usual, unconditional lexicographic preference relations, and corresponds to LP-trees with unconditional local preferences and unconditional importance relation as defined by (Booth et al. 2010).

(Bräuning and Hüllermeyer 2012; Bräuning et al. 2017) extend the expressiveness of LP-trees by allowing to label a node with a set of attributes, considered as a single high-dimensional attribute: the rules in the conditional preference table of the node define orders on the Cartesian product of the domains of its attributes. Note that any (strict) preference relation can in principle be represented by such a LP-tree –



(a) A LP-tree.

(b) A linear LP-tree

(c) A 2-LP-tree

Figure 1: Different LP-trees and the preference relations they induce.

possibly by labelling the root with a set that contains all attributes and an order over the full domain $\underline{\mathcal{X}}$. Generally, we restrict this expressivity by fixing the maximum number of attributes labelling a node. We denote k-LP-tree the trees whose nodes are labelled by at most k attributes. Figure 1c shows a 2-LP-tree whose preference relation cannot be expressed with a regular LP-tree.

3 Learning a preference relation from sales histories

The probabilistic model

As exposed in the introduction, the input of the learning process is a sales history, i.e. a multiset $\mathcal{H} \subseteq \underline{\mathcal{X}}$. Its elements are positive examples, outcomes corresponding to products that may have been, for instance, configured by users of some configurator and bought. Each user has a preference relation $\breve{\succ}$ among products, and is free to configure the products according to her preference. However, for some reasons like the influence of advertisements, she may end up with a product which is not her most preferred one.Yet, the higher an outcome is ranked in the user's preference, the greater the probability that she ends up with it.

Formally, *our idea is to consider that there is a ground, hidden preference relation $\breve{\succ}$, and an associated probability distribution $p_{\breve{\succ}}$ over $\underline{\mathcal{X}}$ which is decreasing and monotonic w.r.t. $\breve{\succ}$*, i.e. if $\mathbf{o} \breve{\succ} \mathbf{o}'$ then $p_{\breve{\succ}}(\mathbf{o}) \geq p_{\breve{\succ}}(\mathbf{o}')$. We consider that $\mathcal{H}$ *is a set of outcomes that have been drawn according to the distribution $p_{\breve{\succ}}$*. In this paper, we study the problem of learning a LP-tree $\mathcal{L}$ which explains $\mathcal{H}$.

In practice, we can split the learning set $\mathcal{H}$ into several clusters and thus reduce the variability of the preferences inside each cluster. Even though each cluster includes the preference of multiple users, they have similar preferences, which can be represented with a unique ground preference relation, with $p_{\breve{\succ}}$ also accounting for the variations between the preferences of these users.

The ranking loss

In order to evaluate our learning algorithms, we must measure how good the learnt preference relation $\succ$ is w.r.t. the hidden preference relation $\breve{\succ}$.

The two distances mostly used to compare preference relations are the Kendall distance and the Spearman distance. These distances penalise as much a rank error on preferred items or on least preferred items.But for recommendation purposes, identifying the preference relation on the preferred items is more useful than identifying it on the least preferred ones.

In order to have a relevant measure, we introduce the notion of *ranking loss*, defined as the normalized difference between the expected ranks of the two relations according to the ground probability $p_{\breve{\succ}}$:

$$\mathsf{rloss}_{p_{\breve{\succ}}}(\breve{\succ}, \succ) = \frac{1}{|\underline{\mathcal{X}}|}\left(E_{p_{\breve{\succ}}}[\mathsf{rank}(\succ, \cdot)] - E_{p_{\breve{\succ}}}[\mathsf{rank}(\breve{\succ}, \cdot)]\right)$$

It is also equal to the mean of the rank differences for all items, weighted by their probabilities of appearance, and normalized by the maximum rank:

$$\mathsf{rloss}_{p_{\breve{\succ}}}(\breve{\succ}, \succ) = \frac{1}{|\underline{\mathcal{X}}|}\sum_{\mathbf{o}\in\underline{\mathcal{X}}} p_{\breve{\succ}}(\mathbf{o})(\mathsf{rank}(\succ, \mathbf{o}) - \mathsf{rank}(\breve{\succ}, \mathbf{o}))$$

Proposition 1. *Let $\breve{\succ}$ and $\succ$ be two preference relations and $p_{\breve{\succ}}$ a probability distribution decreasing w.r.t. $\breve{\succ}$. Then* $0 \leq \mathsf{rloss}_{p_{\breve{\succ}}}(\breve{\succ}, \succ) < 1$. *Furthermore, if $p_{\breve{\succ}}$ is strictly decreasing w.r.t. $\breve{\succ}$, then* $\mathsf{rloss}_{p_{\breve{\succ}}}(\breve{\succ}, \succ) = 0$ *iff* $\breve{\succ} = \succ$.

Proof. All the ranks belong to $\{1, \ldots, |\mathcal{X}|\}$, so $(\mathsf{rank}(\succ, \mathbf{o}) - \mathsf{rank}(\breve{\succ}, \mathbf{o}))/|\mathcal{X}| < 1$ for every $\mathbf{o}$, thus $\mathsf{rloss}_{p_{\breve{\succ}}}(\breve{\succ}, \succ) < 1$. The other two properties follow from the rearrangement inequality (Hardy, Littlewood, and Pólya 1952, sect. 10.2): given two sequences of real numbers $r_1 \leq \ldots \leq r_N$ and $p_1 \geq \ldots \geq p_N$, it holds that $r_1p_1 + \ldots + r_Np_N \leq r_{\sigma(1)}p_1 + \ldots + r_{\sigma(N)}p_N$ for every permutation σ of $\{1, \ldots, N\}$; if the sequences are strictly increasing / decreasing, the minimum is only attained when σ is the identity. In our case, the r_i's correspond to the ranks of the outcomes as ordered by $\breve{\succ}$, the p_i's correspond to the probabilities and σ is the permutation that results in the ranks corresponding to $\succ$. □

In our learning setting, the target preference is unknown, so we cannot measure the ranking loss of an induced preference. However, since the ranking loss of the induced preference is a sum of the ranks of the outcomes weighted by their probabilities of being drawn, we aim to minimize it by

minimizing the normalized empirical mean rank of a set of positive training examples $\mathcal{H}$ drawn according to $p_{\succ}$:

$$\overline{\mathsf{rank}}(\succ, \mathcal{H}) = \frac{1}{|\underline{\mathcal{X}}|} \frac{1}{|\mathcal{H}|} \sum_{\mathbf{o} \in \mathcal{H}} \mathsf{rank}(\succ, \mathbf{o})$$

Note that this optimization problem does not directly depend on $p_{\succ}$, only on the set of examples. Although $p_{\succ}$ explains why $\mathcal{H}$ does not contain only one outcome – the optimal one according to a hidden target LP-tree – our algorithms below do not directly use it, only through the sample $\mathcal{H}$.

4 A greedy learning algorithm

In this Section, we describe a greedy algorithm that learns a k-LP-tree and approximately minimizes this measure. The approach follows the generic scheme defined by (Booth et al. 2010; Bräuning and Hüllermeyer 2012) to learn LP-trees from examples of pairwise comparisons, but we adapt it to learn from positive examples instead.

Algorithm 1 builds the tree in a greedy way, from the root to the leaves, considering, at every step, the subset of the sales history compatible with the current partial instantiation. Given $\mathbf{u} \in \underline{\mathbf{U}}$ with $\mathbf{U} \subseteq \mathcal{X}$, let

$$\mathcal{H}(\mathbf{u}) = \{\mathbf{o} \in \mathcal{H} \mid \mathbf{o}[\mathbf{U}] = \mathbf{u}\}$$

denote the set of outcomes in $\mathcal{H}$ that extend $\mathbf{u}$. Then, at a given currently unlabelled node N (initially, the root node), line 3 considers the set $\mathcal{H}(\mathsf{inst}(N))$ of outcomes in the sales history that are compatible with the assignments made in the branch between the root and N. Function chooseAttributes returns the set of attributes and the CPT that will label the node. The tree is then expanded below N according to the set of labels returned by function generateLabels.

Algorithm 1: Learn a k-LP-tree from a sample $\mathcal{H}$

Input: $\mathcal{X}$, a set of outcomes $\mathcal{H}$ over $\mathcal{X}$
Output: $\mathcal{L}$ the learnt k-LP-tree
Algorithm `LearnLPTree`$(\mathcal{X}, \mathcal{H})$

1 $\mathcal{L} \leftarrow$ unlabelled root node
2 **while** $\mathcal{L}$ *contains some unlabelled node* N **do**
3 $(\mathbf{X}, table) \leftarrow \mathsf{ChooseAttributes}(N)$
4 label N with attributes $\mathbf{X}$ and CPT $table$
5 $L \leftarrow \mathsf{GenerateLabels}(N, \mathbf{X})$
6 **for each** $l \in L$ **do** add new unlabelled node to $\mathcal{L}$, attached to N with edge labelled with l
7 **return** $\mathcal{L}$

Choice of node labels

Given a node N, function chooseAttribute returns an attribute and a CPT so as to explain as well as possible the set of outcomes: we want to induce a LP-tree that ranks as high as possible the elements of $\mathcal{H}(\mathbf{n})$, where $\mathbf{n} = \mathsf{inst}(N)$.

Suppose first that we already know that N must be labelled with attribute set $\mathbf{X}$. Our algorithm learns a CPT that consists of a single, unconditional preference rule.[1] It is not difficult to see that the values of $\underline{\mathbf{X}}$ should be ordered according to their number of occurrences in $\mathcal{H}(\mathbf{n})$ in order to minimize the empirical rank of $\mathcal{H}$.

Example 2. Consider two attributes A and B with respective domain $\{a, a', a''\}$ and $\{b, b'\}$. Assume that $\mathcal{H}$ contains 45 outcomes distributed as follows:

ab	ab'	$a'b$	$a'b'$	$a''b$	$a''b'$
10	9	8	7	6	5

Suppose that we have chosen attribute A to label the root of an induced LP-tree, then the associated CPT should contain the order $a > a' > a''$ over $\underline{A}$, since a has 19 occurrences in $\mathcal{H}$, a' has 15 occurrences, and a'' has 11 occurrences.

In order to decide which attribute, or set of attributes, should label N, consider two attributes $X, Y \notin \mathsf{Anc}(N)$, and suppose that $\{X\}$ is chosen for N. Then Y will appear below X in the induced LP-tree, and will be deemed less important. Let x^* and y^* be the values for X and Y respectively that have the most occurrences in $\mathcal{H}(\mathbf{n})$; then, according to the semantics of LP-trees, for every values $x' \in \underline{X}$, $x' \neq x^*$, and $y' \in \underline{Y}$, $y' \neq y^*$, every outcome that extends $\mathbf{n}x^*y'$ will be considered to be preferred over every outcome that extends $\mathbf{n}x'y^*$. Since $\mathcal{H}$ is assumed to be representative of the preference relation, it should be the case that $|\mathcal{H}(\mathbf{n}x^*y')| > |\mathcal{H}(\mathbf{n}x'y^*)|$ for every pair $(x', y') \in (\underline{X} \setminus \{x^*\}) \times (\underline{Y} \setminus \{y^*\})$; thus the inequality should hold if we take averages as well:

$$\frac{\sum_{y' \in \underline{Y} \setminus \{y^*\}} |\mathcal{H}(\mathbf{n}x^*y')|}{|\underline{Y}| - 1} > \frac{\sum_{x' \in \underline{X} \setminus \{x^*\}} |\mathcal{H}(x'y^*)|}{|\underline{X}| - 1}$$

If the reverse inequality holds, then it cannot be the case that X is more important than Y, and X should not label N.

Example 3. Consider again the dataset of example 2:

$$\frac{|\mathcal{H}(ab')|}{|\underline{B}| - 1} = 9/1 > (8 + 6)/2 = \frac{|\mathcal{H}(a'b)| + |\mathcal{H}(a''b)|}{|\underline{A}| - 1}$$

therefore A is chosen for the root. The resulting LP-tree correctly orders the outcomes according to their numbers of occurrences in $\mathcal{H}$.

We can generalize the approach to sets of attributes. Consider two sets of attributes $\mathbf{X}$ and $\mathbf{Y}$ that do not contain any attribute from $\mathsf{Anc}(N)$ and that are not included into one another. Suppose that N is labelled with $\mathbf{X}$, the associated CPT can order the instantiations of $\mathbf{X}$ according to their number of occurrences in $\mathcal{H}(\mathbf{n})$. Let $\mathbf{U} = \mathbf{X} \setminus \mathbf{Y}$ be the set of variables that are in $\mathbf{X}$ and not in $\mathbf{Y}$ and let $\mathbf{V} = \mathbf{Y} \setminus \mathbf{X}$. Let $\mathbf{x}^*$ (resp. $\mathbf{y}^*$) be the most frequent instantiation of $\mathbf{X}$ (resp. $\mathbf{Y}$) in $\mathcal{H}$. Then for every $\mathbf{x}' \in \underline{\mathbf{X}} \setminus \{\mathbf{x}^*\}$, $\mathbf{v}', \mathbf{v}'' \in \underline{\mathbf{V}}$, every outcome that extends $\mathbf{n}\mathbf{x}^*\mathbf{v}'$ will be preferred to every outcome that extends $\mathbf{n}\mathbf{x}'\mathbf{v}''$. In particular, if $\mathbf{y}^*$ is the most frequent instantiation of $\mathbf{Y}$ in $\mathcal{H}$, let $\mathbf{v}'' = \mathbf{y}^*[\mathbf{V}]$, then for

[1] This is not restrictive, since function chooseAttribute that we describe below separates the values of an attribute into several branches as long as there are enough examples to induce meaningful orders over the domains of subsequent attributes.

every instantiation $\mathbf{u}'$ of $\mathbf{U}$ which is not compatible with $\mathbf{x}^*$, if $\mathbf{x}' = \mathbf{u}' \cdot \mathbf{y}^*[\mathbf{X}]$, it should be the case that outcomes that extend $\mathbf{n}\mathbf{x}^*\mathbf{v}'$ are more frequent in $\mathcal{H}$ than those that extend $\mathbf{n}\mathbf{x}'\mathbf{v}'' = \mathbf{n}\mathbf{u}'\mathbf{y}^*$. Hence it can be shown that we should have

$$\frac{\sum_{\mathbf{v}' \in \underline{\mathbf{V}} \setminus \{y^*[\mathbf{V}]\}} |\mathcal{H}(\mathbf{n}\mathbf{x}^*\mathbf{v}')|}{|\underline{\mathbf{V}}| - 1} > \frac{\sum_{u' \in \underline{\mathbf{U}} \setminus \{x^*[\mathbf{U}]\}} |\mathcal{H}(\mathbf{n}\mathbf{u}'\mathbf{y}^*)|}{|\underline{\mathbf{U}}| - 1} \quad (1)$$

Note that, since we allow sets of attributes at the nodes of LP-trees, several LP-trees can represent the same relation – the extreme structure being a LP-tree with a single node that contains all the attributes and an order of the 2^n possible combinations of values. LP-tree nodes should not be larger than necessary: we now propose a way to recognise nodes that may be shrunk.

We do not want to build LP-trees with node sizes larger than necessary. We now propose a way to recognise when this may happen.

Suppose that $\mathbf{X} \supset \mathbf{Y}$ and that for every $\mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \underline{\mathbf{X}}$ such that $|\mathcal{H}(\mathbf{n}\mathbf{x})| > |\mathcal{H}(\mathbf{n}\mathbf{x}')| > |\mathcal{H}(\mathbf{n}\mathbf{x}'')|$ and $\mathbf{x}[\mathbf{Y}] = \mathbf{x}''[\mathbf{Y}]$ we also have $\mathbf{x}'[\mathbf{Y}] = \mathbf{x}[\mathbf{Y}] = \mathbf{x}''[\mathbf{Y}]$. Then we say that $\mathbf{Y}$ *decomposes* $\mathbf{X}$ at N (w.r.t. $\mathcal{H}$).

The intuition is that $\mathbf{Y}$ decomposes $\mathbf{X}$ at N (w.r.t. $\mathcal{H}$) if node N labelled with $\mathbf{X}$ could be replaced with a node labelled with $\mathbf{Y}$ just above a node labelled with $\mathbf{X} \setminus \mathbf{Y}$.

Example 4. Consider again the attributes of example 2, and suppose that $>$ orders $\underline{\{A, B\}}$ as follows:

$$ab > ab' > ab'' > a'b > a'b' > a'b''$$

Then $ab > ab' > ab''$ and $ab[A] = ab''[A] = a$, and indeed $ab''[A] = ab[A] = ab'[A]$; similarly, $a'b > a'b' > a'b''$ and $a'b[A] = a'b''[A] = a'$, and indeed $a'b''[A] = a'b[A] = a'b'[A]$. This preference relation could be represented with a tree with a single node containing both variables, but also with a tree with A at the root and B below it.

Note that a k-LP-tree with decomposable nodes can always be transformed into a k-LP-tree without decomposable nodes, by "decomposing" these nodes.

Algorithm 2 enumerates the set $\mathcal{G}$ of all subsets of $\mathcal{X} - \mathsf{Anc}(N)$ with cardinality $\leq k$ for some fixed k, starting the search with some initial $\mathbf{X} \in \mathcal{G}$; every time it encounters a new $\mathbf{Y} \in \mathcal{G}$, it checks whether this $\mathbf{Y}$ proves that $\mathbf{X}$ cannot be the label of N. We say that $\mathbf{Y}$ *disproves* $\mathbf{X}$ if:

- $\mathbf{Y} \supset \mathbf{X}$ and $\mathbf{X}$ does not decompose $\mathbf{Y}$; or
- $\mathbf{Y} \subset \mathbf{X}$ and $\mathbf{Y}$ decomposes $\mathbf{X}$; or
- $\mathbf{Y} \not\supset \mathbf{X}$ and $\mathbf{Y} \not\subset \mathbf{X}$ and the reverse of (1) holds.

The next result validates our method for choosing attributes, as it shows that, given enough examples, our algorithm correctly identifies a target LP-tree:

Proposition 2. *Consider a hidden k-LP-tree $\breve{\mathcal{L}}$ with non-decomposable nodes only. Let $\mathcal{H}$ be a multiset of outcomes such that the empirical distribution $p_{\mathcal{H}}$, defined by $p_{\mathcal{H}}(\mathbf{o}) = |\mathcal{H}(\mathbf{o})|/|\mathcal{H}|$, is strictly decreasing w.r.t. $\succ_{\breve{\mathcal{L}}}$. If Algorithm 1 uses Algorithm 2 for* chooseAttributes *and if* generateLabels *always returns $\underline{\mathbf{X}}$ when called for attribute $\mathbf{X}$, then the returned k-LP-tree exactly represents $\succ_{\breve{\mathcal{L}}}$.*

Algorithm 2: Choose an attribute and its CPT

Input: $\mathcal{X}$, node N, a set of outcomes $\mathcal{H}$ over $\mathcal{X}$, k the maximal number of attributes per label
Output: A set of attributes $\mathbf{X}$ and a preference table T
Algorithm `ChooseAttributes`$(\mathcal{X}, N, \mathcal{H}, k)$

```
G ← the set of attributes sets of size at most k
n ← inst(N) ; X ← any set of attributes of G
x* ← argmax_{x∈X} |H(xn)|
for each Y ∈ G \ {X} do
    y* ← argmax_{y∈Y} |H(yn)|
    if Y disproves X then X ← Y ; x* ← y*
> ← an order of X according to the counts {|H(xn)|}_{x∈X}
return X, >
```

Proof. We will look for the root node label; the same reasoning applies to other nodes. Several k-LP-trees can usually correspond to a same relation. We will look for the label of the k-LP-tree $\breve{\mathcal{L}}$ whose root node, labelled by $\mathbf{X}$, is not decomposable.

Let $\mathbf{Y}_i$ be a set of i attributes and $\mathbf{Y}_j$ a set of j attributes different from $\mathbf{Y}_i$. Let $i, j \in [1, k], i \leq j$ and let compare $\mathbf{Y}_i$ and $\mathbf{Y}_j$. There are two possibilities:

First case, suppose that $\mathbf{Y}_i \not\subset \mathbf{Y}_j$. We can use inequality (1) to determine which one is not the root.

Second case, suppose that $\mathbf{Y}_i \subset \mathbf{Y}_j$. We can't compare $\mathbf{Y}_i$ and $\mathbf{Y}_j$ with inequality (1). There are two possibilities : either $\mathbf{Y}_i$ decomposes $\mathbf{Y}_j$ or not. If $\mathbf{Y}_i$ decomposes $\mathbf{Y}_j$, then $\mathbf{Y}_j$ is disproved because $\mathbf{X}$ is not decomposable. If $\mathbf{Y}_i$ doesn't decompose $\mathbf{Y}_j$, then $\mathbf{Y}_i$ is disproved. Indeed, if $\mathbf{Y}_i$ were the root label, it would decompose $\mathbf{Y}_j$.

Ultimately, the correct, non-decomposable set of attributes $\mathbf{X}$ will eventually appear in the enumeration performed by chooseAttributes; it is the only one that disproves, and is not disproved by, all other candidates. □

Managing the split

If function GenerateLabels always returns a blank label, then every node in the tree built by Algorithm 1 will have one child only, and the LP-tree produced will be a linear one. In this case, the algorithm returns a LP-tree that minimizes the empirical expected rank:

Proposition 3. *Let $\mathcal{X}$ be a set of binary attributes. Algorithm 1, when restricted to linear LP-trees, always returns the linear tree $\mathcal{L}$ that minimizes the empirical expected rank* $\overline{\mathsf{rank}}(\succ_{\mathcal{L}}, \mathcal{H})$.

Proof. Given some linear LP-tree $\mathcal{L}$, let $X(\mathcal{L}, i)$ be the variable at depth i and $x^*(\mathcal{L}, i)$ its preferred value. Then

$$\mathsf{rank}(\mathcal{L}, \mathbf{o}) = 2^n - \sum_{i=1}^{n} 2^{n-i} \delta(\mathcal{L}, i, \mathbf{o})$$

where $\delta(\mathcal{L}, i, \mathbf{o}) = 1$ if $\mathbf{o}[X(\mathcal{L}, i)] = x^*(\mathcal{L}, i)$, 0 otherwise (Lang, Mengin, and Xia 2012). Thus:

$$\begin{aligned}
&\operatorname*{argmin}_{\mathcal{L}} \frac{1}{|\mathcal{H}|} \sum_{\mathbf{o}\in\mathcal{H}} \mathsf{rank}(\mathcal{L},\mathbf{o}) = \operatorname*{argmin}_{\mathcal{L}} \sum_{\mathbf{o}\in\mathcal{H}} \mathsf{rank}(\mathcal{L},\mathbf{o}) \\
&= \operatorname*{argmin}_{\mathcal{L}} \sum_{\mathbf{o}\in\mathcal{H}} \left(2^n - \sum_{i=1}^{n} 2^{n-i}\delta(\mathcal{L},i,\mathbf{o})\right) \\
&= \operatorname*{argmax}_{\mathcal{L}} \sum_{i=1}^{n} 2^{n-i} \sum_{\mathbf{o}\in\mathcal{H}} \delta(\mathcal{L},i,\mathbf{o}) \\
&= \operatorname*{argmax}_{\mathcal{L}} \sum_{i=1}^{n} 2^{n-i} |\mathcal{H}(x^*(\mathcal{L},i))|
\end{aligned}$$

A linear LP-tree $\mathcal{L}^*$ that maximises this sum must have, at every level i, $x^*(\mathcal{L}^*,i) = \operatorname{argmax}_{x\in\underline{X}} |\mathcal{H}(x)|$. And the rearrangement inequality indicates that the variables must be ordered in $\mathcal{L}^*$ in order of decreasing counts $|\mathcal{H}(x^*(\mathcal{L},i))|$, likewise Algorithm 1. □

Learning unconditional lexicographic preferences may be too restrictive, but we do not want to learn fully conditional LP-trees, which would have a size exponential in the number of attributes. One way to learn conditional LP-trees without an exponential number of examples is to consider the number of examples that can be used to continue to induce the tree below a given node N: if there is a child N' of N such that $|\mathcal{H}(\mathsf{inst}(N'))|$ is below a fixed threshold τ, then we can decide not to split; if $H(N')$ contains too few examples, it will not be useful anyway to induce the corresponding part of the LP-tree. We use a function GenerateLabels that implements this approach.

Using this threshold τ, the number of branches of the k-LP-tree learnt is limited because each branch corresponds to at least τ examples. So there are at most $\frac{|\mathcal{H}|}{\tau}$ branches and $\frac{n|\mathcal{H}|}{\tau}$ nodes. In Algorithm 2, at node N, one must count the occurrences in $\mathcal{H}(\mathsf{inst}(N))$ of every value of every set of at most k attributes; this can be done in $O(\binom{n}{k} d^k |\mathcal{H}|)$, where d is the maximum domain size of the attributes. The decomposition check can be done in $O(d^k|\mathcal{H}|)$. Furthermore, Algorithm 2 is called for each node. The overall time complexity of the learning algorithm is then $O(n\binom{n}{k} d^{2k}|\mathcal{H}|^2/\tau)$.

Pruning

Algorithm 1 returns a tree $\mathcal{L}$ that approximately minimizes the empirical expected rank $\overline{\mathsf{rank}}(\succ_{\mathcal{L}},\mathcal{H})$ – within the class of acceptable trees, defined by the chosen threshold. This can lead to overfitting. A standard method to address this problem is to use a score function that trades off, for a given tree, a measure of how well the tree fits to the data with a measure of the complexity of the tree: small models tend to have better generalization properties. We define: $S_\phi(\mathcal{H},\mathcal{L}) = -|\mathcal{H}| \cdot \overline{\mathsf{rank}}(\succ_{\mathcal{L}},\mathcal{H}) - |\mathcal{L}| \cdot \phi(|\mathcal{H}|)$ where $|\mathcal{L}|$ denotes the size of the model, i.e. the number of its nodes, and ϕ is a penalty function. In our experiments, we choose $\phi(|\mathcal{H}|) = c$, for c a real constant.

Once a LP-tree has been learnt, it can be pruned in order to improve its ϕ-score. This procedure is executed in a bottom-up fashion. For each non-leaf node N of $\mathcal{L}$, we check whether this node has several outgoing edges. If so, each edge is redirected on the child of N associated to the preferred value of X. If the score is improved, we keep this pruned LP-tree; otherwise, we restore the original outgoing edges.

5 Experimental study on randomly generated data

The first experiments[2] are run on randomly generated, regular LP-trees to verify how well a hidden LP-tree can be rediscovered.

LP-tree generation

We randomly generated complete LP-trees ($k = 1$) with 10 attributes, each having 2 to 6 values (picked with uniform distribution). Each tree was generated in a top-down fashion, starting at the root: at each non-leaf node N, an attribute not already chosen above is picked at random, as well as an order of its values; with probability σ, where σ is a *split coefficient*, N has as many children as X has values; with probability $1-\sigma$, N has a single child.

Given such a randomly picked LP-tree $\breve{\mathcal{L}}$, we draw a fixed number of outcomes for $\mathcal{H}$ using a truncated geometric distribution: the probability of drawing an outcome o of rank r in $\breve{\mathcal{L}}$ is $p_\mu(r) = K\mu(1-\mu)^{r-1}$, where $1/\mu$ is the mean of p_μ.

In the experiments described below, we used a split coefficient $\sigma = 0.2$ and μ was chosen such that the expected value of the drawn rank is $\underline{\mathcal{X}}/4$: this value (which depends on the sizes that have been picked for the attributes) ensures that, if the root attribute has 2 values, then about 88% of the outcomes are in the preferred subtree.

Results

We compared the three variants of the algorithm — LP-Learning without pruning, LP-Learning with pruning and linear LP-Learning — in terms of ranking loss between the hidden and the learnt LP-trees, of size of the learnt tree, and of CPU time necessary to learn the tree (including the pruning when used). The splitting threshold was set to $\tau = 20$ and the penalty function parameter c set equal to 1. Figure 2 shows the results[3] for different sample sizes. Each point represents the mean value on 750 hidden LP-trees.

Figure 2 shows a very good ranking loss. Notice that the ranking loss of linear LP-trees seems to reach a threshold: the hidden LP-trees are not linear, hence the relations they represent are hard to capture with linear structures.

The size of the induced trees increases with the sample size when no pruning is performed, but this size is significantly reduced by the pruning. This, together with the fact that the ranking loss of the pruned LP-trees is always better than that of the unpruned LP-trees, suggest an overfitting

[2] The algorithms have been implemented in Java and have been run on a computer with 8 GB of RAM, a single core 3.4 GHz. Code available at https://github.com/PFGimenez/PhD.

[3] Because the cardinality of $\underline{\mathcal{X}}$ can be huge, the ranking loss is estimated with a Monte-Carlo sampling with 100,000 samples.

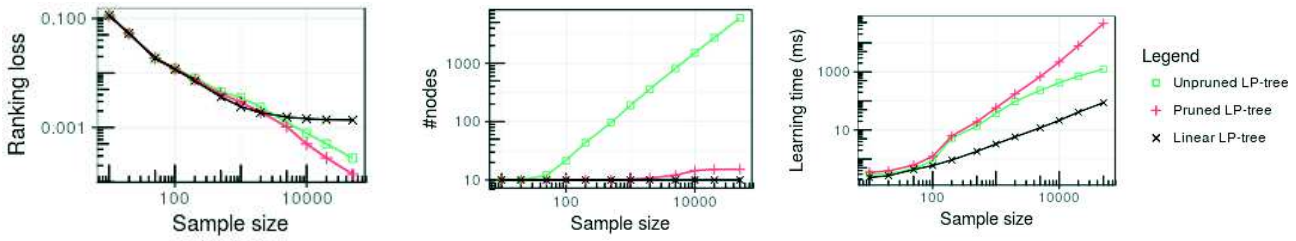


Figure 2: Ranking loss (left), size of the learnt LP-tree (middle) and learning time (right) w.r.t. the sample size.

of the pre-pruning phase. The pruning reduces the size of the learnt LP-tree and enhances its precision, at the cost of a increase in CPU time – that remains sustainable though.

6 Application to value recommendation in product configuration

The experiments described in this Section are based on a real world problem, car configuration, and the experimental results are drawn for a real world data set, namely a sales history provided by Renault, the French car manufacturer.[4] The idea is to use the sales history to build one or several LP-trees, representing the preferences of the past users (the attributes in the LP-tree are the configuration variables). These trees can then be used to recommend values during future configurations: given a partial configuration (a partial instantiation $\mathbf{u} \in \underline{\mathbf{U}}$), the LP-tree builds in polynomial time (Lang, Mengin, and Xia 2012) the best (according to the preference relation it represents) outcome $\mathbf{o}$ compatible with $\mathbf{u}$, (i.e. $\mathbf{o}[\mathbf{U}] = \mathbf{u}[\mathbf{U}]$). The system then recommends the value $\mathbf{o}[X]$, i.e. the value of X in the preferred completion of $\mathbf{u}$.

Dataset and clustering

We use a dataset that is a genuine sales history containing vehicles of the same type sold by Renault, the French car manufacturer. The vehicles of this data set are described by 48 attributes (mostly binary) and the history contains 27088 items.

This sales history correspond to many customers and there is not enough data to learn the preference relation of each customer (an individual does not often buy a car). However, we can expect clusters of similar users which imply clusters of similar sold products that can be found in the sales history, using some classical k-means algorithm.We can then learn a LP-tree for each cluster, that should approximate the preferences of the customers that bought the items in that cluster.

When making a recommendation for a partial outcome $\mathbf{u}$, we use the LP-tree of the cluster that is the nearest to $\mathbf{u}$. This means that, during a configuration session, different LP-trees can be used at different depth in the configuration process. We need to adjust our definition of the empirical mean rank for several LP-trees $\{\mathcal{L}_i\}$ and the orders they represent $\{\succ_i\}$: $\overline{\mathsf{rank}}(\{\succ_i\}, \mathcal{H}) = \frac{1}{|\mathcal{H}|} \sum_{\mathbf{o} \in \mathcal{H}} \min_i \mathsf{rank}(\succ_i, \mathbf{o})$

[4]Datasets available at http://www.irit.fr/~Helene.Fargier/BR4CP/benches.html

Experiment protocol

We use a 10 folds cross-validation protocol. For each item $\mathbf{o}$ in the test set, we simulate a configuration session: we start with an empty item $\mathbf{u}$; for each attribute $X \in \mathcal{X}$, chosen in a random order, the recommender makes a recommendation $\hat{x}$ given $\mathbf{u}$; we then instantiate X with the value $\mathbf{o}[X]$ that has been really chosen by the user; the process is repeated until all attributes have been instantiated. When all items in the test set have been processed, we compute the ratio of accepted recommendations, when the recommended value $\hat{x}$ matched the value $\mathbf{o}[X]$ that had been really chosen.

Results

The learning parameter τ is set equal to 20. The results for the learning of unpruned LP-trees are shown in Figure 3. First, we can remark a positive correlation between the precision rate and the normalised empirical mean rank. This suggests that our theoretical score, the mean rank, is indeed correlated with the experimental precision results.

The precision rate ranges from 78.2% (k=1, 1 cluster) to 88.0% (k=3, 3 clusters); this high precision confirms the practical interest of our approach on real dataset.

The value of the normalised mean rank is in itself interesting, because a random LP-tree would have a normalised mean rank equal to 50% while the LP-trees learnt have a normalised mean rank between 1.2% ($k = 1$, 1 cluster) and $7 \cdot 10^{-7}$% ($k = 3$, 3 clusters).

The precision is maximum for 3 and 4 clusters. While having several clusters increases the expressiveness, each cluster contains fewer examples; this explains the drop in precision with 5 clusters. With more attributes per node (the parameter k) the expressivity is increased, hence a positive impact on the precision.

The size of the model induced (which is the sum of the sizes of the LP-trees learnt for each cluster) seems globally independent of the number of clusters. The effect of the maximum number of attributes per node is stunning, from 2300 nodes with $k = 1$ to 90 nodes with $k = 3$. Pruning did not seem to improve results on this dataset.

7 Conclusion and perspectives

This paper constitutes a first approach to the general problem of learning an ordinal preference model from positive examples (and not from pairwise comparisons). We plan to experiment our approach on more real-world datasets. We also plan to study the sample complexity of our algorithms

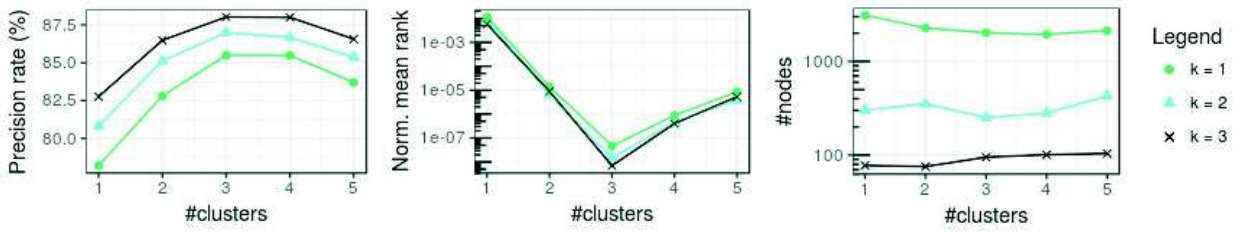


Figure 3: Precision (left), normalised expected rank (middle) and size (right) of learnt unpruned LP-trees w.r.t. the number of clusters and the maximum number of attributes per node k.

in a PAC setting since the experiments suggest that the ranking loss is inversely proportional to the sample size. Another lead is the analysis of the computational complexity of finding a LP-tree minimizing the empirical mean rank. In parallel, searching for a scalable and practical algorithm of exact optimization is worth investigating. We also wish to extend the approach to other ordinal preference models like CP-nets; or, more generally, an extension of this framework to partial orders.

References

Booth, R.; Chevaleyre, Y.; Lang, J.; Mengin, J.; and Sombattheera, C. 2010. Learning conditionally lexicographic preference relations. In *Proceedings of ECAI'10*, 269–274.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. 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 Hüllermeyer, E. 2012. Learning conditional lexicographic preference trees. In Fürnkranz, J., and Hüllermeyer, E., eds., *Proceedings of ECAI'12 Workshop*.

Bräuning, M.; Hüllermeier, E.; Keller, T.; and Glaum, M. 2017. Lexicographic preferences for predictive modeling of human decision making: A new machine learning method with an application in accounting. *European Journal of Operational Research* 258(1):295–306.

Braziunas, D. 2005. Local utility elicitation in GAI models. In *Proceedings of UAI'05*, 42–49.

Brewka, G., et al. 2006. An efficient upper approximation for conditional preference. In *Proceedings of ECAI'06*, volume 141, 472.

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

Flach, P. A., and Matsubara, E. T. 2007. A simple lexicographic ranker and probability estimator. In *Proceedings of ECML'07*, volume 4701, 575–582.

Fraser, N. M. 1993. Applications of preference trees. In *Proceedings of SMC'93*, 132–136.

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

Freund, Y.; Iyer, R. D.; Schapire, R. E.; and Singer, Y. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research* 4:933–969.

Fürnkranz, J., and Hüllermeier, H., eds. 2011. *Preference learning*. Springer.

Gigerenzer, G., and Goldstein, D. G. 1996. Reasoning the fast and frugal way: Models of bounded rationality. *Psychological Review* 103(4):650–669.

Gonzales, C., and Perny, P. 2004. GAI networks for utility elicitation. In *Proceedings of KR'04*, 224–234.

Hardy, G. H.; Littlewood, J. E.; and Pólya, G. 1952. *Inequalities*. Cambridge university press, 2nd edition.

Huffman, C., and Kahn, B. E. 1998. Variety for sale: Mass customization or mass confusion? *Journal of retailing* 74(4):491–513.

Joachims, T. 2002. Optimizing search engines using clickthrough data. In *Proceedings of SIGKDD'02*, 133–142.

Koriche, F., and Zanuttini, B. 2009. Learning conditional preference networks with queries. In *Proceedings of IJCAI'09*, 1930–1935.

Lang, J.; Mengin, J.; and Xia, L. 2012. Aggregating conditionally lexicographic preferences on multi-issue domains. In *Principles and Practice of Constraint Programming*, 973–987. Springer.

Liu, X., and Truszczynski, M. 2015. Learning partial lexicographic preference trees over combinatorial domains. In *Proceedings of AAAI'15*, volume 15, 1539–1545.

Schiex, T.; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI'95*, 631–637.

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

Viappiani, P.; Faltings, B.; and Pu, P. 2006. Preference-based search using example-critiquing with suggestions. *Journal of Artificial Intelligence Research* 27:465–503.

Wallace, R. J., and Wilson, N. 2009. Conditional lexicographic orders in constraint satisfaction problems. *Annals of Operations Research* 171(1):3–25.

Yaman, F.; Walsh, T. J.; Littman, M. L.; and Desjardins, M. 2008. Democratic approximation of lexicographic preference models. In *Proceedings of ICML'08*, 1200–1207.