Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

An Efficient Algorithm To Compute Distance Between Lexicographic Preference Trees

Minyi Li and **Borhan Kazimipour**
Monash University, Australia
minyi.li@monash.edu, borhan.kazimipour@monash.edu

Abstract

Very often, we have to look into multiple agents' preferences, and compare or aggregate them. In this paper, we consider the well-known model, namely, lexicographic preference trees (LP-trees), for representing agents' preferences in combinatorial domains. We tackle the problem of calculating the dissimilarity/distance between agents' LP-trees. We propose an algorithm `LpDis` to compute the number of disagreed pairwise preferences between agents by traversing their LP-trees. The proposed algorithm is computationally efficient and allows agents to have different attribute importance structures and preference dependencies.

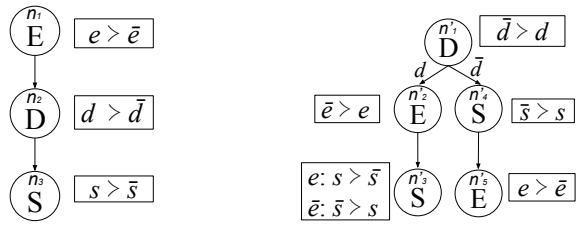
1 Introduction

There are many situations where we have to represent and reason about different agents' preferences over a set of possibly interrelated attributes or variables [Lang *et al.*, 2012; Horniaček, 2008]. For example, one may need to group customers based on their preferences to discover common decision patterns, and recommend actions [Bräuning *et al.*, 2017]. Several work has been published on clustering and predicting preference/ranking data where the ranking over the training labels (correspond to alternative outcomes) is assumed to be given [Hüllermeier *et al.*, 2008; Vembu and Gärtner, 2011; Grbovic *et al.*, 2013; Todorovski *et al.*, 2002]. The task then involves the prediction of strict label order relations. This is, however, impractical when a combinatorial domain is involved: i) data collection becomes infeasible when there is a large number of decision attributes. For instance, even with 5 binary attributes, there will be $2^5 = 32$ unique outcomes. If we collect the preference data via pairwise comparison query, there will be $\binom{32}{2} = 496$ pairs of outcomes that each user needs to compare. ii) predicting ranking is considered as a complex learning task, as the predicted output is a ranking relation rather than single values like those in traditional ML tasks. More generally, with a combinatorial domain, the number of alternatives grows exponentially fast with the number of attributes. In such case, the output of the algorithm is a huge ranking list over the alternative space. Moreover, in tasks like clustering, it is more meaningful to build explanatory models so that one can examine the intra-cluster attribute preferences and attribute importance structures to discover common decision patterns among agents.

In this paper, we consider *lexicographic preference trees* (LP-trees) [Wilson, 2006; Booth *et al.*, 2011] for representing agents' preferences over combinatorial domains. LP-trees are one of the most intuitive and compact representations that correspond to many real-world preferences exhibited by human decision makers [Schmitt and Martignon, 2006; Yaman *et al.*, 2008; Wilson, 2014; Liu and Truszczynski, 2015]. They have a variety of potential applications in the current ML era. For example, they can be used as inputs to cluster agents based on agents' preferences, or to predict the satisfaction level of an agent based on similar agents' LP-trees. However, in common ML practices like clustering or prediction, many existing algorithms rely on the assumption that there is a numerical space they can model. For instance, the famous KNN classification [Cover and Hart, 1967] and K-Means clustering algorithms [Hartigan and Wong, 1979] cannot directly classify or cluster LP-trees. They need to measure the distance between any pair of samples while LP-trees do not provide such information per se. Therefore, some transformations are required to extract distance information between LP-trees. Otherwise, many existing ML algorithms are inapplicable.

In this paper, we tackle the problem of calculating the number of pairwise disagreements (commonly called *Kendall tau ranking distance*) between agents' preferences represented by LP-trees. We propose a novel algorithm, `LpDis`, to compute the distance between two agents' LP-trees without the need of explicitly generating all possible pairs. It avoids any unnecessary comparisons while preserving the accuracy. Moreover, `LpDis` is very flexible: it neither poses any restriction on the structures of LP-trees, nor requires agents to share common preference dependencies or importance structures. Experimental results show that `LpDis` performs very fast and scales well compared to an exhaustive querying method.

In the next section, we provide the preliminaries of LP-trees. We then present the theoretical basis and our algorithm in Section 3 and 4, respectively. At last, we discuss the experimental results in Section 5 and close with conclusions and a brief account of future work in Section 6.



(a) an LP-tree $\mathcal{T}$ (b) another LP-tree $\mathcal{T}'$

Figure 1: LP-tree Examples

2 Lexicographic Preference And LP-trees

Let $\mathcal{V} = \{X_1, \cdots, X_N\}$ be a set of N attributes and D_X be the domain of an attribute X. When X is binary, we write $D_X = \{x, \bar{x}\}$. Let $\mathcal{O}$ be the outcome space, i.e., the Cartesian product of attribute domains $\prod_{X \in \mathcal{V}} D_X$. We denote outcomes as lowercase Greek letters (α, β, etc.). An outcome $\alpha \in \mathcal{O}$ is represented as a vector $\alpha = (x_1, \cdots, x_N)$ where $x_i \in D_{X_i}$.

For any subset of attributes $\mathbf{X} \subseteq \mathcal{V}$, we write $D_{\mathbf{X}} = \prod_{X \in \mathbf{X}} D_X$. We denote the projection of an outcome α on $\mathbf{X}$ as $\alpha[\mathbf{X}]$. For convenience, if $\mathbf{X} = \{X\}$ is a single attribute, we write $\alpha[X]$ instead of $\alpha[\{X\}]$.

Definition 1 (Consistent assignments)**.** *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be the value assignments to two attribute subsets* $\mathbf{X}$ *and* $\mathbf{Y}$*, respectively. We call,* $\mathbf{x}$ *is consistent with* $\mathbf{y}$*, written as* $\mathbf{x} \bowtie \mathbf{y}$*, if they assign the same values to all common attributes* $\mathbf{X} \cap \mathbf{Y}$ *(i.e.,* $\mathbf{x}[\mathbf{X} \cap \mathbf{Y}] = \mathbf{y}[\mathbf{X} \cap \mathbf{Y}]$*); Otherwise, they are conflicting* $\mathbf{x} \not\bowtie \mathbf{y}$*, i.e.,* $\exists X \in \mathbf{X} \cap \mathbf{Y}$ *such that* $\mathbf{x}[X] \neq \mathbf{y}[X]$.

Obviously, if $\mathbf{Y} \cap \mathbf{X} = \emptyset$, we have $\mathbf{x} \bowtie \mathbf{y}$. Also, when $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{x} \bowtie \mathbf{y}$, we have $\mathbf{x}[\mathbf{Y}] = \mathbf{y}$.

A lexicographic order on $\mathcal{O}$ is a total order induced by two major elements: i) the attribute importance order on $\mathcal{V}$; ii) local preferences on attribute values. It orders a pair of outcomes $\{\alpha, \beta\}$ by looking at the attributes in sequence according to their importance, until we reach an attribute X such that the values of X are different between α and β. α and β are then ordered according to the local preference relation over the values of X. This corresponds to the conditional preference theory discussed in [Wilson, 2004], given the assignment $\mathbf{u}$ to the more important attributes in both α and β, $\alpha \succ \beta$ if and only if $\alpha[X] \succ_{\mathbf{u}} \beta[X]$, irrespective of values assigned to the less important attributes in α and β. In general, both the importance relations between attributes and the attribute preferences can be conditioned on the values of more important attributes. Such lexicographic preference relations can be compactly represented by lexicographic preference trees (LP-trees).

2.1 Lexicographic Preference Trees

According to [Booth *et al.*, 2011], an LP-tree $\mathcal{T}$ is a tree where each node n is labelled with an attribute denoted by V_n. Every non-leaf node has between one to a maximum of $|D_{V_n}|$ outgoing branches, each with a branching constraint $V_n = x$ ($x \in D_{V_n}$). Every attribute appears exactly once on every branch of the tree. The tree structure defines the importance relations over attributes: each branch of the tree specifics a descending order (from root to leaf) over attributes. We use $\triangleright$ to denote the importance relation over attributes to distinguish it from the preference relation $\succ$.

There is a conditional preference table (CPT) associated with each node n in an LP-tree, which is defined as follows. Let $\mathbf{P}$ be the set of parent attributes where the preferences over the values of V_n would condition on. The CPT of n is then composed of the local preferences over D_{V_n} for all valuations of $\mathbf{P}$: $\{\mathbf{p} : x^1 \succ \cdots \succ x^k \,|x^i \in D_{V_n}$ and $\mathbf{p} \in D_{\mathbf{P}}\}$. $\mathbf{p}$ is called a parent context.

Example 1 (LP-trees)**.** Figure 1 illustrates two different LP-trees $\mathcal{T}$ and $\mathcal{T}'$ for a simple conference flight reservation scenario. We label every node as n_i and n'_i in $\mathcal{T}$ and $\mathcal{T}'$ respectively. Consider the following binary-valued attributes:

- E - Early departure. e denotes an early trip (e.g., departing at least 2 days earlier); and $\bar{e}$ is a just-on-time trip;
- D - Day-time flight. d (resp. $\bar{d}$) denotes a day-time (resp. a night-time) flight;
- S - Stop-over. s (resp. $\bar{s}$) means a stop-over (resp. non-stop) flight.

Figure 1a is a simple LP-tree $\mathcal{T}$ where the user considers the departure date (E) as the most important attribute: he prefers an early trip ($e \succ \bar{e}$); followed by the time of flight (D): he likes a day-time flight more than a night-time flight ($d \succ \bar{d}$); and at last the stop-over feature (S): he prefers to have some stop-overs ($s \succ \bar{s}$). Note that in $\mathcal{T}$, the attribute importance and attribute preferences are unconditional.
Figure 1b shows a more general example of LP-tree $\mathcal{T}'$, where the user has both conditional attribute importance and conditional attribute preferences. D is the most important attribute, and the user prefers night-time flight over day-time flight ($\bar{d} \succ d$). Then, conditioned on the value of D:

- If $D = d$, the next most important attribute is E (on n'_2) and $\bar{e} \succ e$. Then the least important variable is S (on n'_3), where the CPT of S states that $s \succ \bar{s}$ if $E = e$; otherwise if $E = \bar{e}$, then $\bar{s} \succ s$.
- If $D = \bar{d}$, the second most important attribute is S (on n'_4), and followed by E (on n'_5). For S (resp. E), the CPT states that $\bar{s} \succ s$ (resp. $e \succ \bar{e}$).

An LP-tree $\mathcal{T}$ induces a total order $\succ_{\mathcal{T}}$ on the outcome space $\mathcal{O}$. For a pair of outcomes $\{\alpha, \beta\}$, we query $\mathcal{T}$ in a top-down manner, following the relevant branch according to the common value assignment in α and β. Until we reach a node n in $\mathcal{T}$, where α and β have different values assigned to V_n ($\alpha[V_n] \neq \beta[V_n]$). We look into the CPT of n and conclude that $\alpha \succ_{\mathcal{T}} \beta$ if and only if $\alpha[V_n] \succ^{\mathbf{p}} \beta[V_n]$ where $\mathbf{p}$ is the parent context of V_n specified in α and β ($\mathbf{p} = \alpha[\mathbf{P}] = \beta[\mathbf{P}]$).

Example 2 (Outcome Comparison)**.** Consider the outcome pairs $\alpha = \bar{e}d\bar{s}$, $\beta = eds$, and $\mathcal{T}'$ in Figure 1b, we query $\mathcal{T}'$ accordingly. The first visited node is n'_1 and $V_{n'_1} = D$. As $\alpha[D] = \beta[D] = d$, we continue to move to the left child n'_2, where $V_{n'_2} = E$. Now, as $\alpha[E] \neq \beta[E]$ ($\alpha[E] = \bar{e}$ but $\beta[E] = e$). According to the CPT on n'_2, we have $\bar{e} \succ e$. Therefore, we can directly conclude that $\alpha \succ_{\mathcal{T}'} \beta$.

3 Kendall Tau Distance Between LP-trees

A typical ML scenario is to cluster agents based on their preferences, where we need to look into the similarity or distance between different agents' preferences. In this section, we introduce our novel `LpDis` algorithm that efficiently traverses LP-trees and calculates the number of disagreed pairwise preference relations (Kendall tau ranking distance) between two LP-trees. `LpDis` has the following advantages:

(i) It does not restrict the structures of the LP-trees. The LP-trees can have different preference dependencies between attributes and attribute importance structures;

(ii) It considers general multi-valued domains, instead of restricting to binary-valued attributes;

(iii) It only traverses the nodes in two LP-trees without the need of generating the entire outcome space or querying LP-trees for any outcome pair. Hence, it significantly improves the computation time, which is crucial in large-scale ML tasks.

3.1 Preference Encoded In LP-tree Nodes

As mentioned earlier, an LP-tree $\mathcal{T}$ induces a total order over the outcome space $\mathcal{O}$. This means, for any possible outcome pair $\{\alpha, \beta\}$, it can only be either $\alpha \succ_{\mathcal{T}} \beta$ or $\beta \succ_{\mathcal{T}} \alpha$. Such pairwise preferences are encoded in the nodes of an LP-tree.

Given an LP-tree $\mathcal{T}$ and a node n in $\mathcal{T}$, let $\mathbf{A}$ be the set of ancestor attributes of n, $\mathbf{B}$ be the set of branching attributes (attributes on nodes that have multiple outgoing edges) along the path to n ($\mathbf{B} \subseteq \mathbf{A}$), and $\mathbf{b}$ be branching constraints, i.e., the value assigned to $\mathbf{B}$ along the path to n. Then, n decides the preference order over a pair of outcomes $\{\alpha, \beta\}$ in $\mathcal{T}$ if it is the first node where α and β have different values. We denote the set of all such outcome pairs as $\mathcal{P}_n$. For any of these outcome pairs $\{\alpha, \beta\} \in \mathcal{P}_n$, α and β have the same assignment to $\mathbf{A}$ (and with $\alpha[\mathbf{B}] = \beta[\mathbf{B}] = \mathbf{b}$), but differ on V_n (i.e., attribute on n), and irrespective of the values assigned to the rest of the descendants of n. Formally,

$$\mathcal{P}_n = \big\{\{\alpha, \beta\} \mid \alpha, \beta \in \mathcal{O},\ \alpha[\mathbf{A}] = \beta[\mathbf{A}],\ \alpha[\mathbf{B}] = \beta[\mathbf{B}] = \mathbf{b} \text{ and } \alpha[V_n] \neq \beta[V_n]\big\} \tag{1}$$

Note that $\{\alpha, \beta\}$ and $\{\beta, \alpha\}$ are considered as the same outcome pairs. For any pair of nodes n_i, n_j in an LP-tree, $\mathcal{P}_{n_i} \cap \mathcal{P}_{n_j} = \emptyset$ (i.e., they are disjoint) because every outcome pair $\{\alpha, \beta\}$ is decided at exactly one node in the tree: $\alpha \succ_{\mathcal{T}} \beta$ if and only if the CPT of V_n states that $\alpha[V_n] \succ^{\mathbf{p}} \beta[V_n]$ in the parent context $\mathbf{p}$ included in $\alpha[\mathbf{A}]$ (=$\beta[\mathbf{A}]$). Also, the sum of $|\mathcal{P}_n|$ of all the nodes in an LP-tree, i.e., $\prod_{n \in \mathcal{T}} |\mathcal{P}_n|$, is then equal to the total number of outcome pairs.

Example 3 (Node Outcome Pairs)**.** Consider node n_2 of the LP-tree $\mathcal{T}$ in Figure 1a, where $V_{n_2} = D$. we have $\mathbf{A} = \{E\}$, $\mathbf{B} = \emptyset$. So $\mathcal{P}_{n_2} = \{\{\alpha, \beta\} \mid \alpha[E] = \beta[E] \text{ and } \alpha[D] \neq \beta[D]\}$. Therefore, $\mathcal{P}_{n_2}$ contains the following outcome pairs: $\{eds, e\bar{d}s\}, \{\bar{e}ds, \bar{e}\bar{d}s\}, \{eds, e\bar{d}\bar{s}\}, \{\bar{e}ds, \bar{e}\bar{d}\bar{s}\}, \{ed\bar{s}, e\bar{d}s\}, \{\bar{e}d\bar{s}, \bar{e}\bar{d}s\}, \{ed\bar{s}, e\bar{d}\bar{s}\}, \{\bar{e}d\bar{s}, \bar{e}\bar{d}\bar{s}\}$. Similarly, consider node n_2' in $\mathcal{T}'$ in Figure 1b. We have $V_{n_2'} = E$, $\mathbf{A} = \mathbf{B} = \{D\}$ and $\mathbf{b} = d$. So $\mathcal{P}_{n_2'} = \{\{\alpha, \beta\} \mid \alpha[D] = \beta[D] = d \text{ and } \alpha[E] \neq \beta[E]\} = \{\{\bar{e}ds, eds\}, \{\bar{e}ds, ed\bar{s}\}, \{\bar{e}d\bar{s}, eds\}, \{\bar{e}d\bar{s}, ed\bar{s}\}\}$.

3.2 Node Disagreement Between LP-trees

Since two LP-trees could have different attribute importance structures, for any pair of nodes n, n' in two LP-trees $\mathcal{T}$ and $\mathcal{T}'$, respectively, $\mathcal{P}_n$ and $\mathcal{P}_{n'}$ could contain different sets of outcome pairs. In the following, we define the set of disagreed outcome pairs between nodes in two LP-trees.

Definition 2 (Disagreed outcome pairs between nodes)**.** *Let $\mathcal{T}$, $\mathcal{T}'$ be two LP-trees, n and n' be a node in $\mathcal{T}$ and $\mathcal{T}'$, respectively. The set of disagreed outcome pairs $\mathcal{D}_{n,n'}$ (equivalent to $\mathcal{D}_{n',n}$) is defined as follows:*

$$\mathcal{D}_{n,n'} = \{\{\alpha, \beta\} \in \mathcal{P}_n \cap \mathcal{P}_{n'} \mid \alpha \succ_{\mathcal{T}} \beta \text{ and } \beta \succ_{\mathcal{T}'} \alpha\}$$

Similar to LP-tree that encodes its preferences using two components (attribute preferences and attribute importance), the disagreement between two LP-trees also occurs because of two reasons: different local preferences on attribute values and different structures of attribute importance.

Given two nodes n and n' of the LP-trees $\mathcal{T}$ and $\mathcal{T}'$, respectively, let $\mathbf{A}$ (resp. $\mathbf{A}'$) be the set of ancestor attributes of n (resp. n'), and $\breve{\mathbf{A}}$ be the union of ancestor attributes $\breve{\mathbf{A}} = \mathbf{A} \cup \mathbf{A}'$. Let $\mathbf{B}$ (resp. $\mathbf{B}'$) be the set of branching attributes of n (resp. n'), and $\mathbf{b}$ (resp. $\mathbf{b}'$) be the branching constraints of n (resp. n'). It is important to note that if $\mathbf{b}$ and $\mathbf{b}'$ are conflicting ($\mathbf{b} \not\bowtie \mathbf{b}'$), $\mathcal{P}_n \cap \mathcal{P}_{n'} = \emptyset$, i.e., there exists no possible assignment to the joint ancestor $\breve{\mathbf{A}}$ s.t. both $\mathbf{b}$ and $\mathbf{b}'$ are True. In this case, $|\mathcal{D}_{n,n'}| = 0$. Therefore, when computing the disagreement between pairs of nodes in two LP-trees, we can focus on those where $\mathbf{b} \bowtie \mathbf{b}'$. Let $\breve{\mathbf{B}}$ be the union $\breve{\mathbf{B}} = \mathbf{B} \cup \mathbf{B}'$ and $\breve{\mathbf{b}}$ be the union of branch constraints specified by n and n' (assuming $\mathbf{b} \bowtie \mathbf{b}'$), then the common outcome pairs in $\mathcal{P}_n$ and $\mathcal{P}_{n'}$ would be those that assign the same values to $\breve{\mathbf{A}}$, satisfy the joint constraint $\breve{\mathbf{b}}$ but assigns different values to V_n and $V_{n'}$:

$$\mathcal{P}_n \cap \mathcal{P}_{n'} = \{\{\alpha, \beta\} \mid \alpha[\breve{\mathbf{A}}] = \beta[\breve{\mathbf{A}}],\ \alpha[\breve{\mathbf{B}}] = \beta[\breve{\mathbf{B}}] = \breve{\mathbf{b}},\ \alpha[V_n] \neq \beta[V_n] \text{ and } \alpha[V_{n'}] \neq \beta[V_{n'}]\} \tag{2}$$

Node Disagreement Caused By Local Preference

If n and n' are both labeled with the same attribute ($V_n = V_{n'}$), the disagreement between n and n' is caused by the conflicting local preferences over attribute values D_{V_n}. Let $\mathbf{P}$ (resp. $\mathbf{P}'$) be the set of parent attributes of n (resp. n') in $\mathcal{T}$ (resp. $\mathcal{T}'$), and $\breve{\mathbf{P}} = \mathbf{P} \cup \mathbf{P}'$. Therefore, $\mathcal{D}_{n,n'}$ contains the common outcome pairs where the preference over D_{V_n} are conflicting between n and n' w.r.t. their joint parent context, denoted by $\breve{\mathbf{p}}$. Therefore, we have:

$$\mathcal{D}_{n,n'} = \Big\{\{\alpha, \beta\} \in \mathcal{P}_n \cap \mathcal{P}_{n'} \mid \alpha[V_n] \succ_n^{\breve{\mathbf{p}}} \beta[V_n] \text{ and } \beta[V_n] \succ_{n'}^{\breve{\mathbf{p}}} \alpha[V_n] \text{ where } \breve{\mathbf{p}} = \alpha[\breve{\mathbf{P}}] (= \beta[\breve{\mathbf{P}}])\Big\}$$

To count the number of outcome pairs, let ℓ be the set of pairwise disagreement on attribute values D_{V_n} between n and

n' w.r.t. the possible joint parent context. Note that we only consider a joint parent context $\breve{\mathbf{p}}$ if it is consistent with $\breve{\mathbf{b}}$.

$$\ell = \{x \succ_n^{\breve{\mathbf{p}}} y \text{ and } y \succ_{n'}^{\breve{\mathbf{p}}} x | \, x, y \in D_{V_n},\ \breve{\mathbf{p}} \in D_{\breve{\mathbf{P}}} \text{ and } \breve{\mathbf{p}} \bowtie \breve{\mathbf{b}}\}$$

Besides $\breve{\mathbf{B}}$ and $\breve{\mathbf{P}}$, for any $\{\alpha, \beta\} \in \mathcal{D}_{n,n'}$, α and β must have the same value assigned to the rest of the ancestor attributes in $\breve{\mathbf{A}}$. The number of such possible assignment is $\prod_{X \in \breve{\mathbf{A}} \setminus (\breve{\mathbf{B}} \cup \breve{\mathbf{P}})} |D_X|$. Furthermore, they can have any values assigned to the remaining attributes $\mathcal{V} \setminus (\breve{\mathbf{A}} \cup \{V_n\})$, as any of those attributes is less important than V_n in $\mathcal{T}$ and $V_{n'}$ in $\mathcal{T}'$, respectively. The number of possible combinations of value assignments to these remaining attributes in $\{\alpha, \beta\}$ is $\prod_{Y \in \mathcal{V} \setminus (\breve{\mathbf{A}} \cup \{V_n\})} (|D_Y|)^2$. Therefore, when $V_n = V_{n'}$ we have:

$$|\mathcal{D}_{n,n'}| = |\ell| \times \prod_{X \in \breve{\mathbf{A}} \setminus (\breve{\mathbf{B}} \cup \breve{\mathbf{P}})} |D_X| \times \prod_{Y \in \mathcal{V} \setminus (\breve{\mathbf{A}} \cup \{V_n\})} (|D_Y|)^2 \tag{3}$$

Example 4 (Node Disagreement - Attribute Preference)**.** Consider n_3 in Figure 1a and n_3' in Figure 1b. We have $\mathbf{A} = \mathbf{A}' = \{E, D\} = \breve{\mathbf{A}}$, $\mathbf{B} = \emptyset$, $\mathbf{B}' = \{D\}$ and $\mathbf{b}' = d$, so $\breve{\mathbf{B}} = \{D\}$ and $\breve{\mathbf{b}} = d$. Also, $\mathbf{P} = \emptyset$ and $\mathbf{P}' = \{E\}$ so $\breve{\mathbf{P}} = \{E\}$. Both of the joint parent context e and $\bar{e}$ are consistent with $\breve{\mathbf{b}}$. Then for any $\{\alpha, \beta\} \in \mathcal{P}_{n_3} \cap \mathcal{P}_{n_3'}$, α and β assign the same values to $\{E, D\}$ (either ed or $\bar{e}d$) while assign different values to S. According to the CPT of the two nodes, the local preference on S are conflicting between n_3 and n_3' only when $E = \bar{e}$. Therefore, we have $\ell = \{s \succ_{n_3}^{\bar{e}} \bar{s} \text{ and } \bar{s} \succ_{n_3'}^{\bar{e}} s\}$ and $|\ell| = 1$. $\mathcal{D}_{n_3,n_3'}$ contains only one outcome pair $\{(\bar{e}ds, \bar{e}d\bar{s})\}$, and $\left|\mathcal{D}_{n_3,n_3'}\right| = 1 \times 1 \times 1 = 1$. Note that $\prod_{X \in \emptyset} |D_X| = 1$.

Node Disagreement Caused By Attribute Importance

If n and n' contain different attributes, i.e., $V_n \neq V_{n'}$,

- If $V_{n'}$ is an ancestor attribute of n in $\mathcal{T}$ ($V_{n'} \in \mathbf{A}$), then according to Equation 1, $\mathcal{P}_n$ contains outcome pairs $\{\alpha, \beta\}$ where $\alpha[\mathbf{A}] = \beta[\mathbf{A}]$, and so $\alpha[V_{n'}] = \beta[V_{n'}]$. On the other hand, $\mathcal{P}_{n'}$ contains outcome pairs where the values of $V_{n'}$ differ ($\alpha[V_{n'}] \neq \beta[V_{n'}]$). In which case, $\mathcal{P}_n \cap \mathcal{P}_{n'} = \emptyset$ and $|\mathcal{D}_{n,n'}| = 0$.
- Similarly, if V_n is an ancestor attribute of n' in $\mathcal{T}'$ ($V_n \in \mathbf{A}'$), we will have $|\mathcal{D}_{n,n'}| = 0$.
- Otherwise, if $V_{n'} \notin \mathbf{A}$ and $V_n \notin \mathbf{A}'$, then in the branch of n in $\mathcal{T}$, V_n is more important than $V_{n'}$ ($V_n \triangleright_{\mathcal{T}}^{\mathbf{b}} V_{n'}$). However, in the branch of n' on $\mathcal{T}'$, $V_{n'}$ is more important than V_n ($V_{n'} \triangleright_{\mathcal{T}'}^{\mathbf{b}'} V_n$).

  In such case, the disagreement between n and n' occurs on outcome pair $\{\alpha, \beta\}$ where,

  - For $\mathcal{T}$: α assigns a more preferred value on V_n than β ($\alpha[V_n] \succ_n \beta[V_n]$). So $\alpha \succ_{\mathcal{T}} \beta$.
  - For $\mathcal{T}'$: β assigns a more preferred value on $V_{n'}$ than α ($\beta[V_{n'}] \succ_{n'} \alpha[V_{n'}]$). So $\beta \succ_{\mathcal{T}'} \alpha$

The number of such combinations of values on V_n and $V_{n'}$ is $\binom{|D_{V_n}|}{2} \times \binom{|D_{V_{n'}}|}{2}$. Moreover, there are $\prod_{X \in \breve{\mathbf{A}} \setminus \breve{\mathbf{B}}} |D_X|$ number of possible assignments to joint ancestors, and $\prod_{Y \in \mathcal{V} \setminus (\breve{\mathbf{A}} \cup \{V_n, V_{n'}\})} (|D_Y|)^2$ possible assignments to the remaining variables in the pair $\{\alpha, \beta\}$. Therefore, we have:

$$|\mathcal{D}_{n,n'}| = \binom{|D_{V_n}|}{2} \times \binom{|D_{V_{n'}}|}{2} \times \prod_{X \in \breve{\mathbf{A}} \setminus \breve{\mathbf{B}}} |D_X| \times \prod_{Y \in \mathcal{V} \setminus (\breve{\mathbf{A}} \cup \{V_n, V_{n'}\})} (|D_Y|)^2 \tag{4}$$

Example 5 (Node Disagreement - Attribute Importance)**.** Consider the node n_1 in $\mathcal{T}$ and n_1' in $\mathcal{T}'$ in Figure 1. $V_{n_1} = E$, $V_{n_1'} = D$, $\breve{\mathbf{A}} = \breve{\mathbf{B}} = \emptyset$. Referring to the CPTs, $e \succ_{n_1} \bar{e}$ and $\bar{d} \succ_{n_1'} d$. Therefore, $\forall \{\alpha, \beta\} \in \mathcal{D}_{n_1,n_1'}$, we have $\alpha[E] = e$, $\beta[E] = \bar{e}$ so $\alpha \succ_{\mathcal{T}} \beta$. And $\alpha[D] = d$, $\beta[D] = \bar{d}$ so $\beta \succ_{\mathcal{T}'} \alpha$. Hence, $\mathcal{D}_{n_1,n_1'} = \{\{eds, \bar{e}\bar{d}s\}, \{eds, \bar{e}\bar{d}\bar{s}\}, \{ed\bar{s}, \bar{e}\bar{d}s\}, \{ed\bar{s}, \bar{e}\bar{d}\bar{s}\}\}$. This is consistent with Equation 4: $\left|\mathcal{D}_{n_1,n_1'}\right| = \binom{2}{2} \times \binom{2}{2} \times \prod_{X \in \emptyset} |D_X| \times \prod_{Y \in \mathcal{V} \setminus \{E,D\}} (|D_Y|)^2 = 1 \times 1 \times 1 \times 2^2 = 4$.

4 The Proposed `LpDis` Algorithm

Now, we know how to compute the number of disagreed pairwise preferences on different nodes of two LP-trees. In this section, these results are used as the basis for calculating the Kendall tau distance (the total number of disagreed outcome pairs) between two LP-trees.

Given two LP-trees $\mathcal{T}$ and $\mathcal{T}'$, let n be a node in $\mathcal{T}$, we denote $\mathcal{D}_{n,\mathcal{T}'}$, as the set of disagreed outcome pairs between n and all nodes in $\mathcal{T}'$: $\mathcal{D}_{n,\mathcal{T}'} = \bigcup_{n' \in \mathcal{T}'} \mathcal{D}_{n,n'}$. We also denote $\mathcal{D}_{\mathcal{T},\mathcal{T}'}$ as the set of disagreed outcome pairs between $\mathcal{T}$ and $\mathcal{T}'$: $\mathcal{D}_{\mathcal{T},\mathcal{T}'} = \bigcup_{n \in \mathcal{T}} \mathcal{D}_{n,\mathcal{T}'} = \bigcup_{n \in \mathcal{T}} \left(\bigcup_{n' \in \mathcal{T}} \mathcal{D}_{n,n'} \right)$. Obviously,

$$|\mathcal{D}_{\mathcal{T},\mathcal{T}'}| = \sum_{n \in \mathcal{T}} |\mathcal{D}_{n,\mathcal{T}'}| = \sum_{n \in \mathcal{T}} \sum_{n' \in \mathcal{T}'} |\mathcal{D}_{n,n'}|$$

is the Kendall tau distance.

The proposed `LpDis` algorithm is a nested tree traversal algorithm (see Algorithm 1). Given two LP-trees $\mathcal{T}$ and $\mathcal{T}'$, we traverse $\mathcal{T}$ (could be in any order) (see line 2). For each visited node n in $\mathcal{T}$, we call the function `NodeDis` (line 8–23). `NodeDis` **depth-first** traverses $\mathcal{T}'$ (starting from the root of $\mathcal{T}'$, line 3–4), and calculate the number of disagreed outcome pairs $|\mathcal{D}_{n,\mathcal{T}'}|$ between n and $\mathcal{T}'$, by summing up $|\mathcal{D}_{n,n'}|$ on each visited node $n' \in \mathcal{T}'$. Note that the function `NodeDis` in Algorithm 1 is a recursive implementation of depth-first tree traversal. It traverses $\mathcal{T}'$, updates and accumulates $|\mathcal{D}_{n,n'}|$ into $|\mathcal{D}_{n,\mathcal{T}'}|$ during each recursive call. For each visited node n' in $\mathcal{T}'$:

Algorithm 1: `LpDis` compute Kendall tau distance between two LP-trees

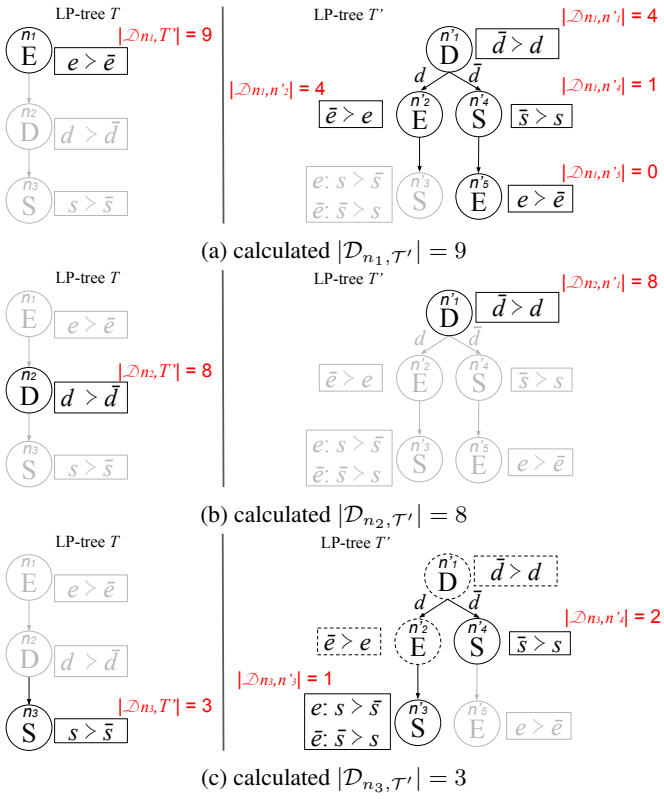
```
1  Algorithm LpDis(T, T')
       Input: two LP-trees T and T'
       Output: |D_{T,T'}|
2      foreach node n ∈ T do
3          n' = root of T'
4          |D_{T,T'}| += NodeDis(n, n', 0)
5      end
6      return |D_{T,T'}|
7
8  Function NodeDis(n, n', |D_{n,T'}|)
       Input: n: a node in T; n': a node in T'; current
              value of |D_{n,T'}|
       Output: |D_{n,T'}|
9      if V_n == V_{n'} then
10         calculate |D_{n,n'}| based on Equation 3
11         |D_{n,T'}| += |D_{n,n'}|
12         return |D_{n,T'}|
13     else
14         if V_{n'} ∉ A then
15             calculate |D_{n,n'}| based on Equation 4
16             |D_{n,T'}| += |D_{n,n'}|
17         end
18         foreach child node n'_c of n' do
19             if b'_c ⋈ b then
20                 |D_{n,T'}| += NodeDis(n, n'_c, |D_{n,T'}|)
21             end
22         end
23     end
```

- If n and n' contain the same attribute $V_n == V_{n'}$ (line 9), we calculate the number of disagreement $|\mathcal{D}_{n,n'}|$ according to Equation 3, add and update the values of $|\mathcal{D}_{n,\mathcal{T}'}|$ (line 10–11). We then return to the upper-level call and prune the rest of the branch (line 12).
- Otherwise (when $V_n \neq V_{n'}$), as mentioned in Section 3.2, if $V_{n'}$ is one of the ancestor attributes of V_n in $\mathcal{T}$ ($V_{n'} \in \mathbf{A}$), then $\mathcal{P}_n \cap \mathcal{P}_{n'} = \emptyset$ and $|\mathcal{D}_{n,n'}| = 0$. Therefore, we only compute $|\mathcal{D}_{n,n'}|$ if $V_{n'} \notin \mathbf{A}$ (line 14). In that case, we calculate $|\mathcal{D}_{n,n'}|$ based on Equation 4, then add and update the value of $|\mathcal{D}_{n,\mathcal{T}'}|$ (line 15–16). Note that we will always have $V_n \notin \mathbf{A}'$ as we stop traversing $\mathcal{T}'$ and return to the upper level call when $V_n == V_{n'}$, as mentioned in the previous bullet point.

  After that, we continue to expand child branches from n'. For each child node n'_c, we examine whether the branching constraint $\mathbf{b}'_c$ of n'_c is consistent with that of n (i.e., $\mathbf{b}$). Conflicting branches are pruned, and for each consistent branch n'_c, we call `NodeDis` with n, n'_c and the updated value of $|\mathcal{D}_{n,\mathcal{T}'}|$ (line 18 – 22).

After ending all recursive calls, `NodeDis` returns $|\mathcal{D}_{n,\mathcal{T}'}|$ to its parent function `LpDis`. We then add $|\mathcal{D}_{n,\mathcal{T}'}|$ to $|\mathcal{D}_{\mathcal{T},\mathcal{T}'}|$ (line 4). After repeating it for each node in $\mathcal{T}$, `LpDis` then returns $|\mathcal{D}_{\mathcal{T},\mathcal{T}'}|$ as the final output (line 6).

Theorem 1. *The proposed `LpDis` algorithm is complete, i.e., the final output is equal to the total number of disagreed outcome pairs between $\mathcal{T}$ and $\mathcal{T}'$.*



(a) calculated $|\mathcal{D}_{n_1,\mathcal{T}'}| = 9$

(b) calculated $|\mathcal{D}_{n_2,\mathcal{T}'}| = 8$

(c) calculated $|\mathcal{D}_{n_3,\mathcal{T}'}| = 3$

Figure 2: Illustration

Proof. During the execution of `LpDis`, we cover every node n in $\mathcal{T}$. For each node n we traverse every node n' in $\mathcal{T}'$ and accumulate $|\mathcal{D}_{n,n'}|$, except the followings:

- If $V_n = V_{n'}$, we prune all child branches from n'. In this case, for any child n'_c of n', we will have V_n ($= V_{n'}$) as an ancestor attribute of n'_c in $\mathcal{T}'$. As discussed in Section 3.2, $\mathcal{P}_n \cap \mathcal{P}_{n'_c} = \emptyset$ and $\left|\mathcal{D}_{n,n'_c}\right| = 0$.
- Similarly, if $V_{n'}$ is an ancestor of V_n in $\mathcal{T}$, we do not consider $|\mathcal{D}_{n,n'}|$, as $\mathcal{P}_n \cap \mathcal{P}_{n'} = \emptyset$ and $|\mathcal{D}_{n,n'}| = 0$.
- If a child node n'_c of n' conflicts with the branching constraints of n, as also discussed in Section 3.2 we will also have $\mathcal{P}_n \cap \mathcal{P}_{n'} = \emptyset$ and $|\mathcal{D}_{n,n'}| = 0$.

In other words, for all possible prunings occur in `LpDis`, we prove that $|\mathcal{D}_{n,n'}| = 0$. And therefore, the final returned output is equal to $\sum_{n\in\mathcal{T}} \left(\sum_{n'\in\mathcal{T}} |\mathcal{D}_{n,n'}| \right) = |\mathcal{D}_{\mathcal{T},\mathcal{T}'}|$. □

Theorem 2. *The computation time of the proposed `LpDis` algorithm is polynomial, quadratic, in the size of the input, i.e., the size of the two LP-trees.*

Proof. For a node n in an LP-tree $\mathcal{T}$, let d_n be the domain size of V_n and p_n be the number of rows in the CPT of n. We can consider the size of an LP-tree as the sum

of the sizes of the CPT on each node of the LP-tree. Assume an LP-tree $\mathcal{T}$ has j nodes and $\mathcal{T}'$ has k nodes, we denote s (resp. s') as the size of $\mathcal{T}$ (resp. $\mathcal{T}'$), then $s = \prod_{i\in\{1,\cdots,j\}} d_{n_i} p_{n_i}$ and $s' = \prod_{i\in\{1,\cdots,k\}} d_{n'_i} p_{n'_i}$. During the execution of `LpDis`, we do a nested tree traverse of $\mathcal{T}$ and $\mathcal{T}'$, so the running time of node visiting is $j \cdot k$. For each pair of nodes n and n' when $V_n \neq V_{n'}$, we can directly calculate $|\mathcal{D}_{n,n'}|$ with Equation 4. Otherwise, if $V_n = V_{n'}$, we will have to count the number of pairwise disagreements on attribute values in every possible joint parent context, the worst case running time is $d_n p_n \cdot d_{n'} p_{n'}$. Therefore, the worst case complexity of `LpDis` is $O(j \cdot k \cdot d_n p_n \cdot d_{n'} p_{n'}) = O(j \cdot d_n p_n \cdot k \cdot d_{n'} p_{n'}) = O(s \cdot s')$. □

Essentially, the running time of `LpDis` depends on the complexity of the LP-trees, i.e., whether they have complex importance structures (with many different branches), and whether the preference on an attribute depends on many other attributes. In the worst case, the number of nodes in an LP-tree and the number of parent context of an attribute grow exponentially in the number of attributes. In which case, however, it is not meaningful to use LP-trees to represent agents' preferences as it constructs a full tree. This is then equivalent to giving out the full ranking over the alternative space.

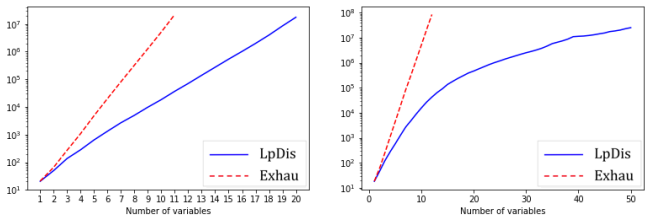
4.1 Illustration

We now demonstrate the execution of `LpDis` with the two LP-trees $\mathcal{T}$ (Figure 1a) and $\mathcal{T}'$ (Figure 1b). `LpDis` traverses $\mathcal{T}$, calculates and accumulates $|\mathcal{D}_{n,\mathcal{T}'}|$ for every node n. Figure 2 shows the calculation for each node.

For n_1 in $\mathcal{T}$ (Figure 2a), $V_{n_1} = E$, we depth-first traverse $\mathcal{T}'$:

- Starting from n'_1 in $\mathcal{T}'$, as demonstrated in Example 5, $|\mathcal{D}_{n_1,n'_1}| = 4$. We update the value of $|\mathcal{D}_{n_1,\mathcal{T}'}|$ as $0 + |\mathcal{D}_{n_1,n'_1}| = 4$. Since there is no branching constraint for n_1 in $\mathcal{T}$, all child branches of n'_1 are consistent with n_1. Therefore, we proceed to each child of n'_1.
- We now reach n'_2 in $\mathcal{T}'$. As $V_{n'_2} = E = V_{n_1}$, we calculate $|\mathcal{D}_{n_1,n'_2}|$ based on Equation 3. For n_1: $\mathbf{A} = \mathbf{B} = \mathbf{P} = \emptyset$, and for n'_2: $\mathbf{A}' = \mathbf{B}' = \{D\}$, $\mathbf{P} = \emptyset$, so $\breve{\mathbf{A}} = \breve{\mathbf{B}} = \{D\}$, $\breve{\mathbf{b}} = d$ and $\breve{\mathbf{P}} = \emptyset$. Referring to the CPT of n_1 and n'_2: $e \succ_{n_1} \bar{e}$, however, $\bar{e} \succ_{n'_2} e$. Hence, the number of disagreed pairwise preference on values of V_{n_1} (= $V_{n'_2}$) is $|\ell| = 1$. Therefore: $|\mathcal{D}_{n_1,n'_2}| = 1 \times \prod_{X\in\emptyset} |D_X| \times \prod_{Y\in\mathcal{V}\setminus(\{D\}\cup\{E\})} (|D_Y|)^2 = 1 \times 1 \times 2^2 = 4$.

  We subsequently end the current recursive call, return to the parent node n'_1 without visiting n'_3, and proceed to another call with parameters n_1, child node n'_4, and the current value $|\mathcal{D}_{n_1,\mathcal{T}}| = 4 + |\mathcal{D}_{n_1,n'_2}| = 8$.
- Similarly for n'_4, $V_{n'_4} = S \neq V_{n_1}$, $\breve{\mathbf{A}} = \breve{\mathbf{B}} = \{D\}$, therefore, according to Equation 4, $|\mathcal{D}_{n_1,n'_4}| = 1$, and the current value $|\mathcal{D}_{n_1,\mathcal{T}}| = 8 + |\mathcal{D}_{n_1,n'_4}| = 9$.



(a) General LP-trees (b) LP-trees with constraints

Figure 3: Running time comparison (in μs; log-scaled; the blue curve and red dashed curve plot the running time of `LpDis` and `Exhau`, respectively)

- At the end, we reach the leaf node n'_5, where $V_{n'_5} = E = V_n$. As they both have the same local preference on E ($e \succ \bar{e}$), therefore, $|\ell| = 0$ and $|\mathcal{D}_{n_1,n'_5}| = 0$

After that, `NodeDis` returns to its parent function `LpDis` with $|\mathcal{D}_{n_1,\mathcal{T}'}| = 9$. We then update $|\mathcal{D}_{\mathcal{T},\mathcal{T}'}| = |\mathcal{D}_{n_1,\mathcal{T}'}| = 9$.

We then move to n_2 in $\mathcal{T}$, and depth-first traverse $\mathcal{T}'$ (see Figure 2b). Right on the root node n'_1, as $V_{n_2} = V_{n'_1} = D$, $\breve{\mathbf{A}} = \{E\}$, $\breve{\mathbf{B}} = \breve{\mathbf{P}} = \emptyset$, and $|\ell| = 1$ ($d \succ_{n_2} \bar{d}$ but $\bar{d} \succ_{n'_1} d$), we have $|\mathcal{D}_{n_2,n'_1}| = 1 \times 2 \times 2^2 = 8$ (according to Equation 3). We then directly return to `LpDis` with $|\mathcal{D}_{n_2,\mathcal{T}'}| = |\mathcal{D}_{n_2,n'_1}| = 8$, and update $|\mathcal{D}_{\mathcal{T},\mathcal{T}'}| = 9 + |\mathcal{D}_{n_2,\mathcal{T}'}| = 9 + 8 = 17$. All child branches of n'_1 in $\mathcal{T}'$ are pruned in this iteration.

Lastly, for n_3 in $\mathcal{T}$ (Figure 2c), we depth-first traverse $\mathcal{T}'$:

- On n'_1, $V_{n'_1} = D$ is an ancestor attribute of n_3 in $\mathcal{T}$, we proceed to its child nodes without any computation.
- Similarly on n'_2, as $V_{n'_2} = E$ is an ancestor attribute of n_3 in $\mathcal{T}$, we proceed to the next call to the leaf node n'_3.
- On n'_3, as shown previously in Example 4, we have $|\mathcal{D}_{n_3,n'_3}| = 1$. We update $|\mathcal{D}_{n_3,\mathcal{T}'}| = 0 + 1 = 1$. Then, we return to n'_2, and subsequently return to n'_1 and further proceed to its right child n'_4.
- On n'_4, $V_{n'_4} = V_{n_3} = S$, $\breve{\mathbf{A}} = \{E, D\}$, $\breve{\mathbf{B}} = \{D\}$, and $|\ell| = 1$. According to Equation 3, $|\mathcal{D}_{n_3,n'_4}| = 1 \times 2 \times 1 = 2$. So $|\mathcal{D}_{n_3,\mathcal{T}'}| = 1 + |\mathcal{D}_{n_3,n'_4}| = 3$. It then prunes the rest of the branch and return to n'_1. `NodeDis` then returns to `LpDis` with $|\mathcal{D}_{n_3,\mathcal{T}'}| = 3$

After that, $|\mathcal{D}_{\mathcal{T},\mathcal{T}'}| = 17 + |\mathcal{D}_{n_3,\mathcal{T}'}| = 20$. Now, we have completed the traverse of $\mathcal{T}$ and `LpDis` returns $|\mathcal{D}_{\mathcal{T},\mathcal{T}'}| = 20$ as the Kendall tau distance between $\mathcal{T}$ and $\mathcal{T}'$.

5 Experiments

We compare the proposed approach with a baseline technique called `Exhau` which performs an exhaustive query over all possible outcome pairs. We generate random LP-trees by varying the number of attributes, the structure of the trees, and the local attribute preferences.

In the first set of experiments, we consider N (number of variables) between 1 to 20 and run 1000 experiments on each number of attributes. All variables are binary. Figure 3a plots the average time spent by these algorithms against the number

of attributes. We can see that LpDis runs significantly faster than the Exhau algorithm. As the number of attributes grows the gap between LpDis and Exhau increases. When $N = 10$, LpDis performs 1000 times faster than Exhau. While it is infeasible to run large-scale experiments with Exhau when the number of variables is more than 11, on average, LpDis runs on 20 attributes in about 15 seconds.

In the second set of experiments, we limit the maximum number of parents of each attribute to 5, and the maximum number of nodes in an LP-tree to be the square of the number of attributes (N^2). The results are shown in Figure 3b. In this case, LpDis presents only a polynomial growth on the number of attributes. It runs on 50 attributes in about 25 seconds. In contrast, Exhau does not show any noticeable improvements even after introducing the constraints.

6 Conclusion and Future Work

We investigated the problem of computing the number of pairwise disagreements (Kendall tau distance) between two agents' preferences represented by LP-trees. We proposed an algorithm called LpDis to calculate such distance with any classes of LP-trees, where the agents do not necessarily share common preference dependencies or attribute importance structures. LpDis retains the accuracy, yet efficiently computes the Kendall tau distance by traversing the LP-trees without generating or querying any outcome pair. As demonstrated with experiments, the improvement in running time compared to the exhaustive querying method is significant.

This research opens up a couple of interesting topics for future exploration, including but not limited to: i) extension for incomplete LP-trees, where some of the attributes are missing in the LP-tree branches, or some of the conditional preference tables (CPTs) on the nodes of the LP-tree are incomplete. While in this work we assume the LP-trees are complete, human preferences in real-world are often incomplete, and possibly inconsistent. We intend to extend the current formulation to accommodate incomplete and/or inconsistent LP-trees. ii) various ML tasks on LP-trees. It would be interesting to cluster or predict agents' preferences by having LP-trees as training samples. We also plan to apply LP-trees in label ranking problem, to discover common decision-making patterns in a group of agents.

References

[Booth *et al.*, 2011] Richard Booth, Yann Chevaleyre, Jérôme Lang, Jérôme Mengin, and Chattrakul Sombattheera. Learning conditionally lexicographic preference relations. In *Proceedings of the 23rd Benelux Conference on Artificial Intelligence*, pages 371–372, 2011.

[Bräuning *et al.*, 2017] Michael Bräuning, Eyke Hüllermeier, Tobias Keller, and Martin Glaum. 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, 2017.

[Cover and Hart, 1967] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[Grbovic *et al.*, 2013] Mihajlo Grbovic, Nemanja Djuric, Shengbo Guo, and Slobodan Vucetic. Supervised clustering of label ranking data using label preference information. *Machine Learning*, 93(2):191–225, Nov 2013.

[Hartigan and Wong, 1979] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[Horniaček, 2008] Milan Horniaček. Negotiation, preferences over agreements, and the core. *International Journal of Game Theory*, 37(2):235–249, Jun 2008.

[Hüllermeier *et al.*, 2008] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16):1897 – 1916, 2008.

[Lang *et al.*, 2012] Jérôme Lang, Jérôme Mengin, and Lirong Xia. *Aggregating Conditionally Lexicographic Preferences on Multi-issue Domains*, pages 973–987. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[Liu and Truszczynski, 2015] 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.

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

[Todorovski *et al.*, 2002] Ljupco Todorovski, Hendrik Blockeel, and Saso Dzeroski. *Ranking with Predictive Clustering Trees*, pages 444–455. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Vembu and Gärtner, 2011] Shankar Vembu and Thomas Gärtner. *Label Ranking Algorithms: A Survey*, pages 45–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Wilson, 2004] Nic Wilson. Extending cp-nets with stronger conditional preference statements. In *AAAI*, volume 4, pages 735–741, 2004.

[Wilson, 2006] Nic Wilson. An efficient upper approximation for conditional preference. In *Proceedings of the 2006 Conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29 – September 1, 2006, Riva Del Garda, Italy*, pages 472–476, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.

[Wilson, 2014] Nic Wilson. Preference inference based on lexicographic models. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, ECAI'14, pages 921–926, Amsterdam, The Netherlands, The Netherlands, 2014. IOS Press.

[Yaman *et al.*, 2008] Fusun Yaman, Thomas J. Walsh, Michael L. Littman, and Marie desJardins. Democratic approximation of lexicographic preference models. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 1200–1207. ACM, 2008.