The Thirty-Second International Florida
Artificial Intelligence Research Society Conference (FLAIRS-32)

Learning Optimal and Near-Optimal Lexicographic Preference Lists

Ahmed Moussa
School of Computing
University of North Florida
Jacksonville, FL
nagar@aucegypt.edu

Xudong Liu
School of Computing
University of North Florida
Jacksonville, FL
xudong.liu@unf.edu

Abstract

We consider learning problems of an intuitive and concise preference model, called *lexicographic preference lists* (*LP-lists*). Given a set of examples that are pairwise ordinal preferences over a universe of objects built of attributes of discrete values, we want to learn (1) an optimal LP-list that decides the maximum number of these examples, or (2) a near-optimal LP-list that decides as many examples as it can. To this end, we introduce a dynamic programming based algorithm and a genetic algorithm for these two learning problems, respectively. Furthermore, we empirically demonstrate that the sub-optimal models computed by the genetic algorithm very well approximate the de facto optimal models computed by our dynamic programming based algorithm, and that the genetic algorithm outperforms the baseline greedy heuristic with higher accuracy predicting new preferences.

Introduction

Preferences are ubiquitous and important to research fields such as recommender systems, decision making and social sciences. In this work, we study preference relations of objects that are combinations of values in discrete attributes. Preference relations vary and the research community has seen promising preference models such as graphical models such as conditional preference networks (Boutilier et al. 2004), lexicographic preference trees (Booth et al. 2010; Liu and Truszczynski 2013; 2015), and lexicographic preference (Liu and Truszczynski 2016; 2018), and logical models such as penalty logic (De Saint-Cyr, Lang, and Schiex 1994), possibilistic logic (Dubois, Lang, and Prade 1994), and answer set optimization (Brewka, Niemelä, and Truszczynski 2003). We focus on a particular lexicographic preference model, called *lexicographic preference lists*, or *LP-lists* for short, an intuitive and concise model describing the importance ordering of the attributes and the preference orderings of the values in the attributes (Fishburn 1974). In particular, we examine the learning problem of LP-lists for a given description of the set of objects and a given set of examples. This problem is considered in two settings: learning the LP-lists that agree with the maximum, or close to the maximum, number of examples.

Learning optimal LP-lists with maximum number of satisfied examples has been proven NP-hard (Schmitt and Martignon 2006; Liu and Truszczynski 2015). In this paper, we introduce a dynamic programming based approach that learns optimal LP-lists in exponential time in the number of attributes and the size of attribute domains. Despite this exponentiality, it is a considerable reduction from the factorial time complexity of the brute force algorithm that checks every ordering of attributes and every preference order within each attribute domain. We show the optimality of our algorithm, and present experimental results demonstrating its effectiveness for large domains of objects.

Algorithms to learning near-optimal LP-lists have been proposed in the literature, including the greedy heuristic algorithm (Liu and Truszczynski 2018). Local search algorithms have been used to learn other preference models, e.g., tree-structured conditional preference networks (Allen, Siler, and Goldsmith 2017). To the best of our knowledge, learning algorithms based on local search techniques have not been applied to learning LP-lists. In this paper, we propose to learn near-optimal LP-lists using a *genetic algorithm* (Mitchell 1998), one of the promising local search algorithms, where LP-lists are straightforward represented as chromosome strings. We conduct empirical evaluation of our genetic algorithm and compare it with the greedy heuristic and our optimal algorithm. Our results suggest that the genetic algorithm performs very close to our optimal algorithm and outperforms the greedy heuristic by a margin.

In the following, we first formally define what LP-lists are. We then present our optimal algorithm using dynamic programming, as well as our near-optimal genetic algorithm. We will discuss our experimental results before we conclude and point to future work.

Lexicographic Preference Lists

Let us denote by $\mathcal{A} = \{X_1, \ldots, X_n\}$ a finite set of attributes. Each attribute $X_i \in \mathcal{A}$ has a finite domain D_i of values such that $D_i = \{x_{i,1}, \ldots, x_{i,m_1}\}$. The universe $U_\mathcal{A}$ defined by $\mathcal{A}$ is the Cartesian product of the attribute domains $D_1 \times \ldots \times D_n$. We call elements in $U_\mathcal{A}$ *objects*. A *lexicographic preference list* (*LP-list*) over $\mathcal{A}$ is a list of attributes in $\mathcal{A}$, each labeled by a total order over that attribute domain. Attributes in a LP-list are distinct and a subset of $\mathcal{A}$.

Let $L = X_{i_1} \rhd \ldots \rhd X_{i_p}$ be an LP-list over $\mathcal{A}$, and o and o' two objects in $U_{\mathcal{A}}$. We say that o is at least as good as o' in L, denoted $o \succeq_L o'$, if (1) $o(X) = o'(X)$ for all $X \in \mathcal{A}$, or (2) there is $i_j \in [i_1, i_p]$ such that $o(X_{i_j}) \succ o'(X_{i_j})$ and $o(X_q) = o'(X_q)$ for all $q \in [i_1, i_j)$. Then, we say that o is strictly preferred to o', denoted $o \succ_L o'$, if $o \succeq_L o'$ and $o' \not\succeq_L o$, and that o is equivalent to o', denoted $o \approx_L o'$, if $o \succeq_L o'$ and $o' \succeq_L o$.

Accordingly, given two objects o and o, and an LP-list L, objects can be compared by an LP-list as follows. For each attribute X_{i_j} in L, starting from the first one, we check if o has a better (or worse) value on X_{i_j} than o'. If so, we stop and report $o \succ_L o'$ ($o' \succ_L o$, resp.). Otherwise, o and o' having same value on X_{i_j}, we continue to the next attribute. If we finish having checked all attributes, we stop and report $o \approx_L o'$. Therefore, this task is done in linear time in the size of the input.

Consider a universe of vehicles of four attributes: Body-Type (B) with values sedan (s) and truck (t), Color (C) with black (k), white (w) and blue (b), Make (M) with Toyota (t) and Chevrolet (c), and Price (P) with low (l), medium (m) and high (h). An LP-list can be $B \rhd M \rhd C$, where B is labeled by $s \succ t$, M by $t \succ c$, and C by $w \succ b \succ k$. According to this LP-list, we see that a medium-priced blue Toyota sedan is preferred to a low-priced white Chevrolet sedan, and two differently priced black Chevrolet trucks are equivalent.

Dynamic Programming Algorithm

Given a set $\mathcal{E}$ of examples and an attribute $X \in \mathcal{A}$, we want to compute the optimal local preference ordering of X, denoted $LPO(X)$, that satisfies the maximum number of examples in $\mathcal{E}$ just by X alone.

From $\mathcal{E}$ we first build a matrix M where $M_{i,j}$ denotes the number of examples in $\mathcal{E}$ that prefer i to j on attribute X. Examples with same value of X in both are not counted in M, so that $M_{i,j} = 0$ if $i = j$. This first step takes $O(|\mathit{Dom}(X)|^2)$ both space and time. Then, we use M to compute the $LPO(X)$ as follows.

Let $S \subseteq \mathit{Dom}(X)$ be a subset of the domain of X, where $|S| > 1$. We denote by $C(S, \mathcal{E})$ the maximum number of examples in $\mathcal{E}$ that can be satisfied by any total order on S. Thus, we have the following.

$$C(S,\mathcal{E}) = \begin{cases} \max\limits_{i,j \in S}\{M_{i,j}, M_{j,i}\} & \text{if } |S| = 2 \\ \max\limits_{i \in S}\{C(S - \{i\}, \mathcal{E}) + \sum\limits_{j \in S - \{i\}} M_{j,i}\} & \text{if } |S| > 2 \end{cases}$$

Therefore, $LPO(X)$ is the total order on X that satisfies $C(\mathit{Dom}(X), \mathcal{E})$ examples in $\mathcal{E}$. Both $LPO(X)$ and $\mathcal{E}$ are computed using the following procedure in Algorithm 1, a dynamic programming based procedure recording calculated results in tables T and L.

We now analyze the space and time complexity of Algorithm 1 as follows. The space complexity results from the matrix M, and tables L and T. Let $x = |\mathit{Dom}(X)|$ be the number of values in X's domain, and m the number of examples in $\mathcal{E}$. Then, space complexity is $O((2^x) \cdot \frac{x}{2} + x^2 + 2^x) = O(x \cdot 2^x)$. This asymptotic prohibitive space is acceptable, if the size of the attribute's domain is relatively small, often the case in practice.

Algorithm 1: *computeLPO*$(\mathcal{E}, X)$ % computes LPO for an attribute for given examples

```
   Input: $\mathcal{E}$ is the set of example, and $X \in \mathcal{A}$ is an
          attribute in $\mathcal{A}$
   Output: $LPO(X)$
1  Procedure computeLPO($\mathcal{E}$, $X$):
2      Create an empty table $L$ s.t. $L[U]$ is an empty list
        for every $U \subset \mathit{Dom}(X)$;
3      Create matrix $M$ as described earlier;
4      Create an empty table $T$ s.t. $T[U] = 0$ for every
        $U \subset \mathit{Dom}(X)$;
5      Set $T[U]$ and $L[U]$ for $|U| = 2$ according to
        above formula;
6      for $i \leftarrow 3$ to $|\mathit{Dom}(X)|$ do
7          foreach $S \subset \mathit{Dom}(X)$ s.t. $|S| = i$ do
8              $T[S] \leftarrow \max_{i \in S}\{T(S - \{i\}) + \sum_{j \in S - \{i\}} M_{j,i}\}$;
9              $L[S] \leftarrow \arg\max_{(L[S-\{i\}], i): i \in S}\{T(S - \{i\}) + \sum_{j \in S - \{i\}} M_{j,i}\}$;
10         end
11     end
12     return $L[\mathit{Dom}(X)]$;
```

To calculate the time complexity, we examine the algorithm closely. We assume structures M, T, and L are constant time accessible. Lines 2 to 5 take time $O(2^x + m + 2^x + \binom{x}{2})$, respectively. The loop from line 6 to line 11 considers all subsets $S \subset \mathit{Dom}(X)$ and $|S| \geq 3$, for each of which $T[S]$ and $L[S]$ are computed. Each takes time $O(|S| \cdot |S|)$. So this loop takes time $O(\sum_{S \subset \mathit{Dom}(X) \wedge |S| \geq 3} |S|^2)$. Then, we have $\sum_{S \subset \mathit{Dom}(X) \wedge |S| \geq 3} |S|^2 = \binom{x}{3} \cdot 3^2 + \ldots + \binom{x}{x} \cdot x^2 \leq (\binom{x}{3} + \ldots + \binom{x}{x}) \cdot x^2 \leq 2^x \cdot x^2$. Therefore, the time complexity of Algorithm 1 is $O(m + x^2 \cdot 2^x)$. This is a clear reduction from the factorial performance of the brute-force approach that checks all permutations of $\mathit{Dom}(X)$.

Let us denote by $\mathit{Attr}(T)$ the set of attributes in $\mathcal{A}$ labeling the nodes in LPL T, by $\alpha|_T$ the partial obtained from object α restricted to attributes showing up in T. Then, we define $\mathcal{E}|_T = \{(\alpha|_T, \beta|_T) : (\alpha, \beta) \in \mathcal{E} \text{ and } \alpha|_T \neq \beta|_T\}$ to be the multi-set of examples obtained from $\mathcal{E}$ restricted to attributes showing up in T.

We say an LPL is *optimal* to $\mathcal{E}$ if it satisfies the maximum number of examples in $\mathcal{E}$. Inspired by the Held-Karp algorithm (Held and Karp 1962), we see that, if an LPL T is optimal to $\mathcal{E}$, then every T's subtree T' rooted at r is optimal to $\mathcal{E}|_{T'}$. Clearly, this property is true because, were the subtree T' to be not optimal, T could be changed to satisfy more examples by altering the order of $\mathit{Attr}(T')$ in T'.

We devise the Algorithm 2 to learn optimal lexicographic preference lists. It is brute-force enhanced by memorizing the optimal subtrees for all subsets of $\mathcal{A}$.

Let us consider the space and time complexities of Algorithm 2. We let $n = |\mathcal{A}|$ be the number of attributes, and $\bar{x} = \max\{|\mathit{Dom}(X)| : X \in \mathcal{A}\}$ the maximum attribute domain size. As with the space complexity, the algorithm uses

Algorithm 2: *computeLPL*$(\mathcal{E}, \mathcal{A})$ % computes optimal LPL for given examples

```
   Input: $\mathcal{E}$ is the set of example, and $\mathcal{A}$ is the set of attributes
   Output: $LPL$
 1 Procedure computeLPL($\mathcal{E}$, $\mathcal{A}$):
 2     Create an empty table $T$ s.t. $T[U] = 0$ for every $U \subset \mathcal{A}$;
 3     Create an empty table $L$ s.t. $L[U]$ is an empty list for every $U \subset \mathcal{A}$;
 4     Set $T[U]$ and $L[U]$ for $|U| = 1$ using computeLPO in Algorithm 1;
 5     for $i \leftarrow 2$ to $|\mathcal{A}|$ do
 6         foreach $S \subset \mathcal{A}$ s.t. $|S| = i$ do
 7             $T[S] \leftarrow \max_{X \in S} \{T(S - \{X\}) + C(Dom(X), \{(\alpha, \beta) \in \mathcal{E} : \alpha(Y) = \beta(Y) \text{ for all } Y \in S - \{X\}\})\}$;
 8             $L[S] \leftarrow \arg\max_{(L[S-\{X\}], computeLPO(X)) : X \in S} \{T(S - \{X\}) + C(Dom(X), \{(\alpha, \beta) \in \mathcal{E} : \alpha(Y) = \beta(Y) \text{ for all } Y \in S - \{X\}\})\}$;
 9         end
10     end
11     return $L[\mathcal{A}]$;
```

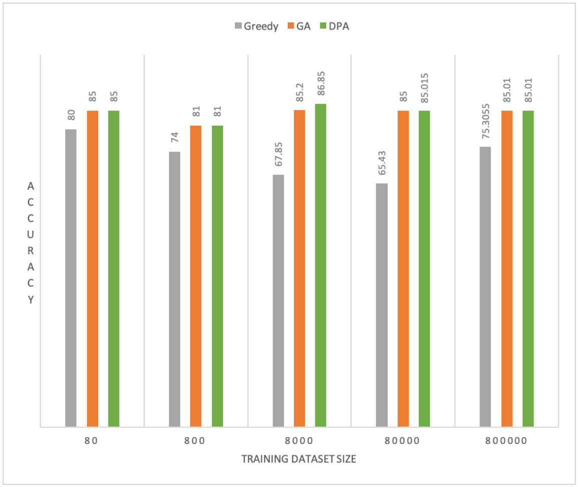

Figure 1: Testing accuracy

tables T and L, and the space designated by the calls to Algorithm 1. The size of L is bounded by $\sum_{0 \le i \le n} \binom{n}{i} \cdot i \cdot \bar{x} \le n \cdot \bar{x} \cdot \sum_{0 \le i \le n} \binom{n}{i} = n \cdot \bar{x} \cdot 2^n$. This gives us a space complexity of $O(2^n + n \cdot \bar{x} \cdot 2^n + \bar{x} \cdot 2^{\bar{x}})$, which is $O(n \cdot \bar{x} \cdot 2^n + \bar{x} \cdot 2^{\bar{x}})$.

Lines 2 to 4 take time $O(2^n + 2^n + n \cdot (m + \bar{x}^2 \cdot 2^{\bar{x}}))$ respectively. The loop from line 5 to line 10 takes time $\sum_{S \subset \mathcal{A} \wedge |S| \ge 2} |S| \cdot (m + \bar{x}^2 \cdot 2^{\bar{x}})$. Therefore, the time complexity of Algorithm 2 is $O(2^n + 2^n + n \cdot (m + \bar{x}^2 \cdot 2^{\bar{x}}) + (m + \bar{x}^2 \cdot 2^{\bar{x}}) \cdot 2^n \cdot n)$, which is $O((m + \bar{x}^2 \cdot 2^{\bar{x}}) \cdot 2^n \cdot n)$.

Genetic Algorithm

The idea of the genetic algorithm is inspired from the natural selection theory in biology. Generally, the population's fitness will increase until a steady state. In such steady state, no improvement can be done once the population has reached this stable state. This state could contain a global or local optimal solution.

Each candidate solution or chromosome is composed of many traits, features, attributes, or simply "genes" with each gene being one value or "allele". For our learning problem of LP-lists, chromosomes are encoded as follows a string of attributes and values in their domains, where upper-case letters represent attributes and lower-case letters represent values the attributes can be of. Taking the previous example of the LP-list in the cars domain: $B \triangleright M \triangleright C$ with the same local preference orderings. Its chromosome representation clearly is "Bst Mtc Cwbk".

Using the fitness function that returns the number of correctly classified examples by the LP-list, we devise the genetic algorithm as follows. Step 1: create 100 random LP-list as initial chromosomes. Step 2: select the top 50 chromosomes overall according to the fitness function and let them produce two children by crossover and mutation respectively. Step 2 is repeated for 100 generations before termination. In step 2, crossover is achieved by shuffling the ordering of the attributes in the chromosomes, and mutation by shuffling the ordering of values of a randomly chosen attribute.

Results

In this section, we present our empirical analysis of our two algorithm: the dynamic programming based algorithm, for which we call *DPA*, and the genetic algorithm, which we short-hand to *GA*.

To evaluate our algorithms DPA and GA, we take the greedy algorithm as a baseline and perform empirical analysis on sets of examples given by hidden randomly generated LP-lists. The examples are produced with a noise percentage of examples that are flipped to create inconsistent examples to simulate practical settings.

Domains of 10 attributes, each of 5 values, are used for our experiments. Thus, the universe contains 5^{10} objects, giving $\binom{5^{10}}{2} \approx 5 \times 10^{13}$ possible examples. We first generate a random LP-list of these attributes with random orderings as their local preferences, and a set D of random examples for training and testing. Then, set D is processed based on a noise percentage $\mathcal{N}$: $|D| \cdot \mathcal{N}$ examples are randomly selected and flipped. We reserve 80% of D to train an LP-list model and the other 20% to test it. Our experiments are for $\mathcal{N}$ of value 15%, and for D of size $10^3, 10^4, \ldots, 10^6$. The instance for every D is repeated 5 times and the average accuracies and computational time are reported as follows.

We see, in Figure 1, that DPA obtains the highest accuracy on the testing examples. GA finishes as a very close second, within 1% compared to DPA. Greedy finishes last. We attribute this to the fact that our GA's stochastic beaming start with multiple LP-lists and generations of improvements of them.

Figure 2 shows the total computational time, including

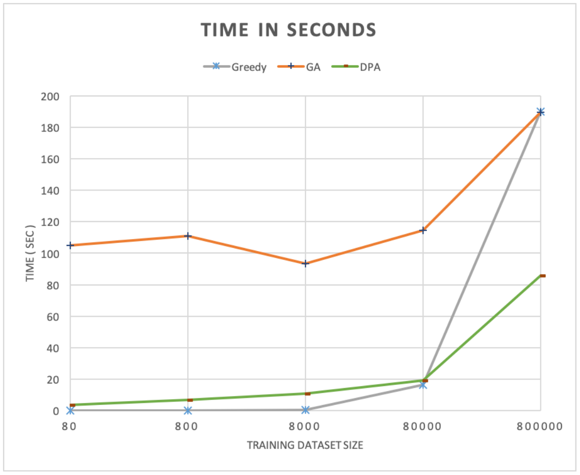

Figure 2: Computational time

both training and testing, for various training data sizes. Clearly, DPA, despite of its exponential time complexity, outperforms GA on all datasets. This is because the computational time of GA accumulates over generations. Greedy takes the least amount of time until the size of the training set picks up to very large. This is attributed to the larger constant in the asymptotic notion of Greedy than that of DPA. GA takes the most time as it goes through generations of the selecting processes.

Conclusion and Future Work

We studied the learning problems of LP-lists, a preference formalism that is intuitive and concise over objects consisting of categorical attributes. We introduced an algorithm DPA that computes optimal LP-lists that decide the most number of given examples. DPA is based on dynamic programming, and it reduces the factorial time complexity of the pure brute force algorithm to exponential, at a cost of exponential space. Besides, we introduced a genetic algorithm GA for computing near-optimal LP-lists that satisfy as many given examples as it can. To evaluate our algorithms, we conducted substantial experiments showing that, for large example sets of sizes up to 10 million over the universe of over 9 million objects, DPA outperforms GA and baseline Greedy in both testing accuracy and computational time, with GA being a very close second in accuracy. For future work, we plan to perform experimental studies on preferential data generated from real-world datasets such as classification and regression datasets in the machine learning community. We also intend to extend our algorithms to allow learning more general lexicographic preference models (Liu and Truszczynski 2015).

References

Allen, T. E.; Siler, C.; and Goldsmith, J. 2017. Learning tree-structured cp-nets with local search. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference*.

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

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

Brewka, G.; Niemelä, I.; and Truszczynski, M. 2003. Answer set optimization. In *IJCAI*, volume 3, 867–872.

De Saint-Cyr, F. D.; Lang, J.; and Schiex, T. 1994. Penalty logic and its link with dempster-shafer theory. In *Uncertainty Proceedings 1994*. Elsevier. 204–211.

Dubois, D.; Lang, J.; and Prade, H. 1994. Possibilistic logic 1.

Fishburn, P. C. 1974. Exceptional paper—lexicographic orders, utilities and decision rules: A survey. *Management science* 20(11):1442–1471.

Held, M., and Karp, R. M. 1962. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics* 10(1):196–210.

Liu, X., and Truszczynski, M. 2013. Aggregating conditionally lexicographic preferences using answer set programming solvers. In *International Conference on Algorithmic Decision Theory*, 244–258. Springer.

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

Liu, X., and Truszczynski, M. 2016. Learning partial lexicographic preference trees and forests over multi-valued attributes. In *the 2nd Global Conference on Artificial Intelligence*, EPiC Series in Computing, 314–328. EasyChair.

Liu, X., and Truszczynski, M. 2018. Preference learning and optimization for partial lexicographic preference forests over combinatorial domains. In *Proceedings of the 10th International Symposium on Foundations of Information and Knowledge Systems (FoIKS)*. Springer.

Mitchell, M. 1998. *An introduction to genetic algorithms*. MIT press.

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