2019

On learning and visualizing lexicographic preference trees

Ahmed S. Moussa
University of North Florida

ON LEARNING AND VISUALIZING LEXICOGRAPHIC PREFERENCE TREES

by

Ahmed S. Moussa

A thesis submitted to the
School of Computing
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
SCHOOL OF COMPUTING

April, 2019

The thesis "On Learning and Visualizing Lexicographic Preference Trees" submitted by Ahmed S. Moussa in partial fulfillment of the requirements for the degree of Master of Science in Computer Science has been

Approved by the thesis committee: Date

Dr. Xudong Liu
Thesis Advisor and Committee Chairperson

Dr. Ayan Dutta

Dr. Sandeep Reddivari

Accepted for the School of Computing:

Dr. Sherif Elfayoumy
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

Dr. William Klostermeyer
Interim Dean of the College

Accepted for the University:

Dr. John Kantner
Dean of the Graduate School

ACKNOWLEDGEMENT

I would like to thank committee members, professors, and friends. I like to express my appreciation and thanks to my advisor Dr. Xudong Liu for the support, comments and guidance throughout the development of this thesis. A special thanks to Dr. Sherif Elfayoumy, School of Computing Director, for the advice and support, while achieving this milestone in life and career.

CONTENTS

FIGURES

TABLES

ABSTRACT

Preferences are very important in research fields such as decision making, recommender systems and marketing. The focus of this thesis is on preferences over combinatorial domains, which are domains of objects configured with categorical attributes. For example, the domain of cars includes car objects that are constructed with values for attributes, such as 'make', 'year', 'model', 'color', 'body type' and 'transmission'. Different values can instantiate an attribute. For instance, values for attribute 'make' can be Honda, Toyota, Tesla or BMW, and attribute 'transmission' can have automatic or manual. To this end, this thesis studies problems on preference visualization and learning for lexicographic preference trees, graphical preference models that often are compact over complex domains of objects built of categorical attributes. Visualizing preferences is essential to provide users with insights into the process of decision making, while learning preferences from data is practically important, as it is ineffective to elicit preference models directly from users.

The results obtained from this thesis are two parts: 1) for preference visualization, a web-based system is created that visualizes various types of lexicographic preference tree models learned by a greedy learning algorithm; 2) for preference learning, a genetic algorithm is designed and implemented, called GA, that learns a restricted type of lexicographic preference tree, called unconditional importance and unconditional preference tree, or UIUP trees for short. Experiments show that GA achieves higher accuracy compared to the greedy algorithm at the cost of more computational time. Moreover, a Dynamic Programming Algorithm (DPA) was devised and implemented that computes an optimal UIUP tree model in the sense that it satisfies as many examples as possible in the dataset. This novel exact algorithm (DPA), was used to evaluate the quality of models computed by GA, and it was found to reduce the factorial time complexity of the brute force algorithm to exponential. The major contribution to the field of machine learning and data mining in this thesis would be the novel learning algorithm (DPA) which is an exact algorithm. DPA learns and finds the best UIUP tree model in the huge search space which classifies accurately the most number of examples in the training dataset; such model is referred to as the optimal model in this thesis. Finally, using datasets produced from randomly generated UIUP trees, this thesis presents experimental results on the performances (e.g., accuracy and computational time) of GA compared to the existent greedy algorithm and DPA.

Chapter 1

INTRODUCTION

The area of preferences has recently gained broad attention from the artificial intelligence research community. For example, the Artificial Intelligence Journal (AIJ) published a special issue in 2011 devoted to preferences titled "*Representing, Processing, and Learning Preferences: Theoretical and Practical Challenges*" [Fürnkranz11]. The AI Magazine also devoted a special issue in 2008 to preferences [Fürnkranz11]. This demonstrates the importance of the field of preferences in AI. Many workshops on preference learning and ranking topics were organized by the machine learning researchers at NIPS 2004 and 2005, ECML/PKDD 2008 and 2009, SIGIR 2008 and 2009 [Fürnkranz11].

Various preference models have been proposed, including: numerical models, such as fuzzy constraint satisfaction problems; logics-based models, such as answer set optimization; and graphical models, such as conditional preference networks and lexicographic preference trees. This thesis focuses on the visualization and learning problems for lexicographic preference trees.

One of the key components of the research on preferences is preference visualization, where preference models of an agent are visualized to help the agent gain insights into their decision-making process. Despite its clear importance, preference visualization has received little attraction from the researchers. A web-based system that visualizes various types of lexicographic preference tree models learned by an existing greedy learning algorithm was developed.

Another critical component is preference learning, referring to the problem of learning preference models of an agent or a group of agents from observations that explicitly or implicitly acquired of the agent(s). The typical goal of preference learning is to generalize the training data into a model to use it later for preference prediction, such as to predict the preferences of a new and similar person or the preferences of the same person but in a new situation. *Learning to rank* is a research area that utilizes preference learning methods with a goal of predicting preferences in a total order form over a set of alternatives which is useful in recommendations. The book "Preference Learning" by Darmstadt Marburg, Johannes Fürnkranz, Eyke Hüllermeier [Fürnkranz11] provides a comprehensive overview with many survey chapters that introduce subfields of preference learning and explain important applications in different areas. In addition, the book tried to structure the field via proposing a unified notation and categorization per each learning task and learning technique, which helps future research in the field. Thus, this thesis explores genetic algorithms and dynamic programming in learning preferences. The below hierarchy in Figure 1 shows where this thesis fits.

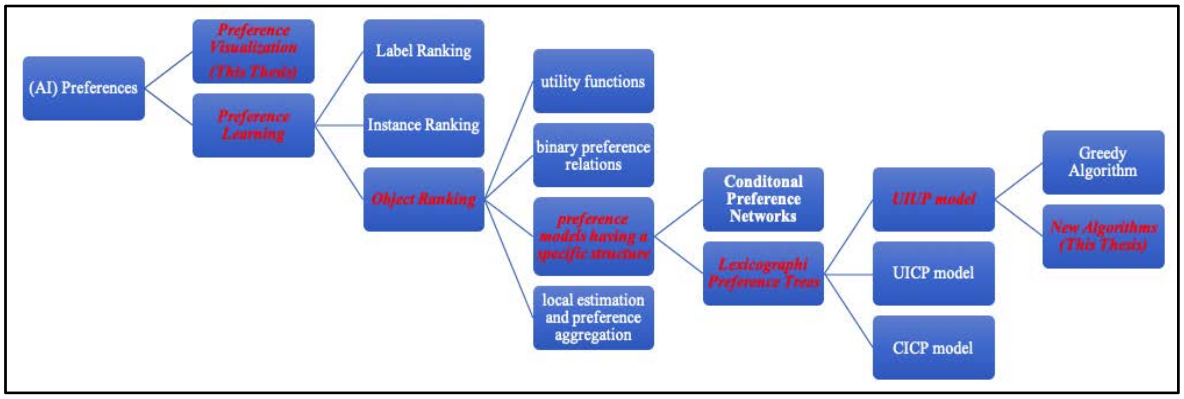

(AI) Preferences
Preference Visualization (This Thesis)
Preference Learning
Label Ranking
Instance Ranking
Object Ranking
utility functions
binary preference relations
preference models having a specific structure
local estimation and preference aggregation
Conditonal Preference Networks
Lexicographi Preference Trees
UIUP model
UICP model
CICP model
Greedy Algorithm
New Algorithms (This Thesis)

Figure 1: Preference Learning Research Tasks and Approaches

The focus of this thesis is on the preference learning problem for combinatorial domains. A combinatorial domain is given by a set of attributes $A = \{ X_1 , X_2 , \ldots , X_n \}$ with each attribute X_i associated with a set of categorical values, the size of which is bounded by a constant. The combinatorial domain *compactly* represents the Cartesian product of attributes in *A*. We call an ordered pair (α, β) an *example*, indicating that the agent prefers object α to β, both objects from the combinatorial domain. The learning problem is framed as follows: Given a set of examples *(E)*, learn a UIUP preference tree model that satisfies as many examples in *E* as possible.

Chapter 2

RELATED WORK

This section discusses the relevant work and information that this thesis is based on. There are major topics related to this research: preferences, genetic algorithms, voting theory, data mining, and graph theory. The key topic to this research is order theory in mathematics and all these topics are related. Related topics are discussed in details and an in-depth analysis is provided. Having a deep understanding of the major topics and concepts above is critical to this research. The shortcoming in techniques in these areas that relate to current problem of preference learning are explained too.

2.1 Lexicographic Preference Trees

Consider the below image that explains some preferences. In Figure 2, ordering all attributes in yellow and ordering all values within each attribute are required.

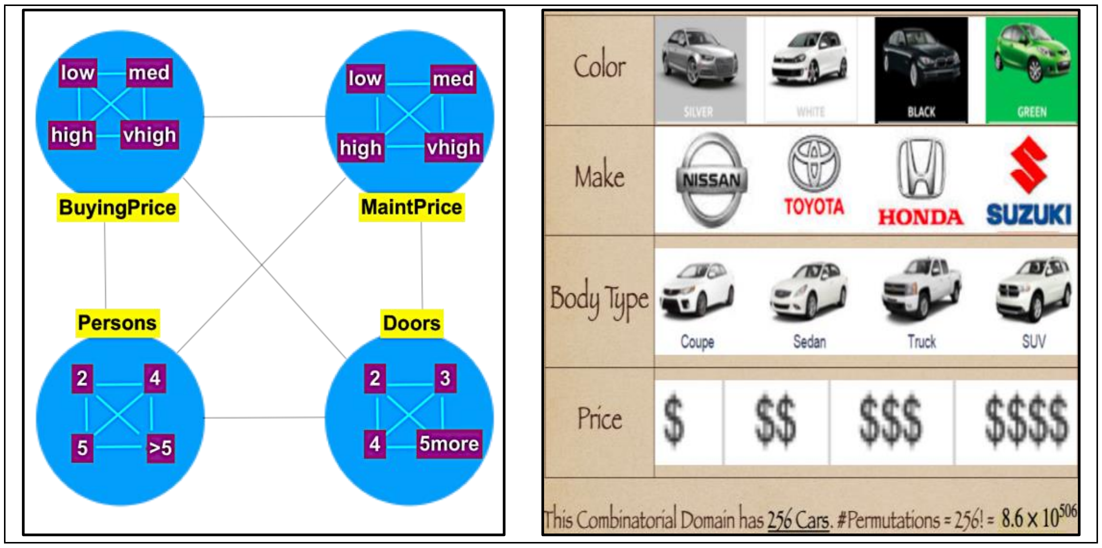


Figure 2: Preferences in Two Car Domain

Challenges:

- The edge weights on the above graph dynamically change and aren't static.
- The above figure is both directional and weighted graph. However, weights and edges change all the time based on what node is visited first. This is because visiting "MaintPrice" node first will remove weights of all dataset's pairs that got correctly or incorrectly classified by assuming the user's first importance / priority is "Maintenance Price" with "low > med > high > vhigh" as preferences.

Assumptions:

- Assume a combinatorial domain where choosing different values for a set of attributes can produce different products.
- Assume the user's preference relation is a linear order.

- Assume the users has a model in mind where the order of attributes matter and the order of values under each attribute is also important.

Here is an example from the cars domain in Figure 2. Order attributes from most important to least important and order values under each attribute from most preferred to least. This car domain is considered a combinatorial domain where 256 different cars (products) could be instantiated. Different agents may order these 256 cars differently per their preferences. There is $256! = 8.6 \times 10^{506}$ different permutations.

2.1.1 Object Ranking Based on Lexicographic Preferences

Consider the problem of learning someone's ordinal preferences in a certain combinatorial domain. Booth et al. authors of "Learning conditionally lexicographic preference relations" introduced Lexicographic Preferences-Trees (LP-tress) as a general graphical representation to model different classes of preference relations [Booth10]. However, there was one important assumption which is assuming "preferences are lexicographic". If someone's preferences are lexicographic then a tree structure can model such order like English words order in a dictionary. Sometimes, preference relations captured from the user contains noise or inconsistency. Such inconsistency or noise is better to be identified and removed because these would not make preferences lexicographic. Assuming lexicographic preferences, LP-trees can capture many different preference relations classes. Different preference relations classes emerge depending on the importance of the order of attributes marking out the nodes in the LP-tree and on the order of local preferences on each attribute / node which could be conditioned on a parent node's value. The UIUP model is the focus of this research. Other classes of LP-trees are identified in the below Table 1. When LP-trees are pruned, they are called PLP-trees or Partial Lexicographic Preferences-Trees.

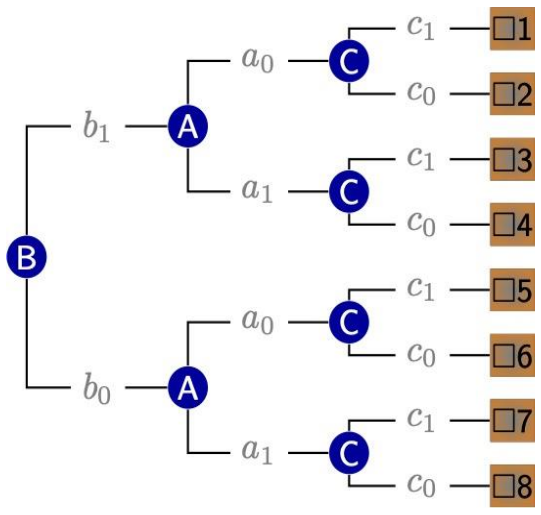
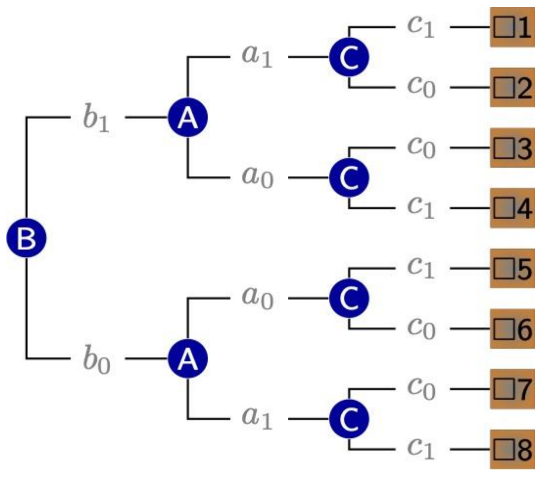
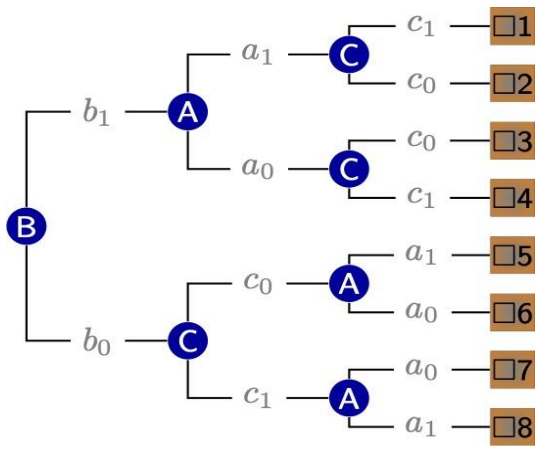
LP-Tree's name	Importance of Attribute	Preference of Values	Example
Unconditional Importance Unconditional Preference (UIUP)	Unconditioned	Unconditioned	 B b_1 b_0 A A a_0 a_1 a_0 a_1 C c_1 c_0 1 2 3 4 5 6 7 8
Unconditional Importance Conditional Preference (UICP)	Unconditioned	Conditioned	 B b_1 b_0 A a_1 a_0 a_0 a_1 C c_1 c_0 c_0 c_1 c_1 c_0 c_0 c_1 1 2 3 4 5 6 7 8
Conditional Importance Conditional Preference (CICP)	Conditioned	Conditioned	 B b_1 b_0 A a_1 a_0 C c_1 c_0 c_0 c_1 C c_0 c_1 A a_1 a_0 a_0 a_1 1 2 3 4 5 6 7 8

Table 1: Example of LP-Trees: UIUP, UICP, CICP

Current algorithms to learn such preferences are greedy and brute-force. There are two greedy algorithms; one for learning LP-trees proposed by Richard Booth [Booth10] and one for learning PLP-trees proposed by Xudong Liu [Liu15]. Both are generic greedy algorithms able to generate LP-trees of various types. For example, the greedy algorithm for learning LP-trees takes a set E of examples, then constructs a tree that satisfies the examples by adding nodes from the root to the leaves. The nodes are added in a greedy approach by picking an attribute that maximizes a certain gain among the remaining E. These current state-of-the-art greedy preference-learning approaches work analogously to the greedy decision tree induction algorithms like Hunt's algorithm, ID3, C4.5, CART, SPRINT in data mining. These greedy algorithms make a series of locally optimum greedy decisions to pick an attribute to partition the remaining data and grow a decision tree. Current greedy approaches aren’t optimal and don’t perform well especially when there is noise in the dataset. The assumed performance evaluation of a preference model is the number of test records correctly predicted by the model. Traditional classification techniques including Support Vector Machines, fail dramatically even with small datasets of size 10 pairs of objects because an object data point can appear both positive and negative relative to the other object data point. Also, preference-learning greedy approach can’t handle noise well. This thesis contributes and proposes more accurate algorithms.

2.2 Preference Learning

The preference learning field is about inducing a predictive model from available data. The model learning technique could be supervised or unsupervised. There are important points to clarify about human preferences. Human preferences are vulnerable to inconsistency which makes the learning problem complex. For example, someone may prefer food A over food B today, but this same person may change preference tomorrow. Inconsistency may appear to reflect real user preferences over time or the user may present insincere preferences either by mistake or intentionally. Inconsistency is a big challenge because there is no model that can predict a conflicting preference. For example, if a user said his preferences are A > B and B > A, then this is like a loop and there is no useful information to learn here. In this section, an overview of preference learning field is provided with the objective of establishing a unified terminology. The focus of this thesis is on learning to rank, which is extensively studied preference learning problem. Genetic algorithms will be introduced to this type of problem and will be used to solve such preference problem in this thesis. Ranking problems can be categorized as follows: label ranking, instance ranking, and object ranking [Fürnkranz11].

Preference elicitation isn't always simple, particularly in complex realistic applications. Hence, preference modeling and representation languages are introduced in literature. However, new learning algorithms for the automatic preference discovery are needed.

Such algorithms are very useful for discovering individual preferences in e-commerce since personalization of products and services are starting to trend nowadays. This was a recent research topic in machine learning, knowledge discovery, and recommender systems. Some of the approaches studied were approximating the utility function and collaborative filtering that estimate a user's preferences from other customers' preferences. Indeed, there are many formalizations for the problems of preference learning based on various settings such as the underlying preference model type or the empirical data type used as an input. In decision theory literature, researchers used two approaches for preference modeling such as utility functions and preference relations; both approaches differ like classification and regression. The later approach predicts complex structures, such as rankings rather than single values [Fürnkranz11]. Like any research field in knowledge representation and reasoning in AI, reasoning with preferences has been recognized as a particularly promising research direction for artificial intelligence (AI). A preference-based problem has the advantage of increased flexibility like preferences are considered relaxed constraint that could be violated.

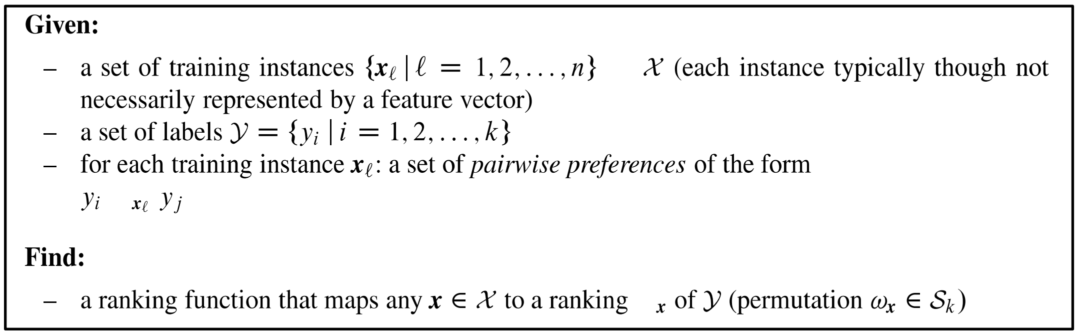
2.2.1 Preference Learning Tasks

"Learning to rank" problem gained the most attention in the machine learning literature in recent years among many problems in the realm of preference learning. Commonly accepted terminology has not yet been established. The book "preference learning" by Johannes Fürnkranz and Eyke Hüllermeier proposed a unifying and clarifying terminology for the most important types of ranking problems [Fürnkranz11]. Generally, preference learning tasks involve learning a function to predict preferences over pairs of items from a set while the required relation form total order. This is relation or function is learned from a training set of items for which preferences are given. This problem type is typically called a ranking problem. The term “ranking” is used frequently in different domains and is used in preference learning too to categorize different problems.

In the field of operations research, the term “ranking” is used for arranging a set of objects in a total order; which has similarities. The common terminology of supervised learning tasks such as classification will be used in this thesis. A data object typically consists of the input and the output or an instance and a class label; also, called predictive and target or independent variable and dependent variable in statistics. One instance normally is represented by a vector of features. Prediction process receives two instances as input and output binary class that indicate whether first instance is ranked higher than the second or not. Some binary class labels could be yes/no higher/lower or even 1/0. Some problem types allow incomparability which means both instances can’t be compared or equally preferred. In this thesis, the focus is on total order so there will always be a binary preference relation. This total-order restriction is used because of the assumption in this thesis that the customer or the user must pick one instance only such as when buying a car or a house from e-commerce websites or even when deciding to marry in which choosing one item only is a necessity and the user can’t afford picking, choosing or buying two items. Different types of ranking problems are examined and discussed in the following paragraphs. There are three types of ranking problems which are label ranking, instance ranking, and object ranking [Fürnkranz11].

2.2.1.1 Label Ranking

The goal in this type of problems is to learn a “label ranker” which assigns labels to an instance from most preferred to least. Such ordering of labels is picked from all permutations of the set of labels. Conventional classification in data mining could be treated as a label ranking problem when instance x maps to only one single class label which is the top. The input to the label ranker will be a training data T in the form of pairwise preferences such as Xi > Xj which means label i is preferred to label j for instance X. Such observations simply consist of an instance and two ordered labels. For example, a picture and two labels suggesting this picture is more likely a dog than a cat or a mammal than a bird or a mammal than a fish. Johannes Fürnkranz and Eyke Hüllermeier formalized the label ranking problem in [Fürnkranz11] as the following:



Given:

- a set of training instances $\{\boldsymbol{x}_\ell \mid \ell = 1, 2, \ldots, n\}$ $\mathcal{X}$ (each instance typically though not necessarily represented by a feature vector)
- a set of labels $\mathcal{Y} = \{y_i \mid i = 1, 2, \ldots, k\}$
- for each training instance $\boldsymbol{x}_\ell$: a set of *pairwise preferences* of the form $y_i \quad _{\boldsymbol{x}_\ell} y_j$

Find:

- a ranking function that maps any $\boldsymbol{x} \in \mathcal{X}$ to a ranking $\quad _{\boldsymbol{x}}$ of $\mathcal{Y}$ (permutation $\omega_{\boldsymbol{x}} \in \mathcal{S}_k$)

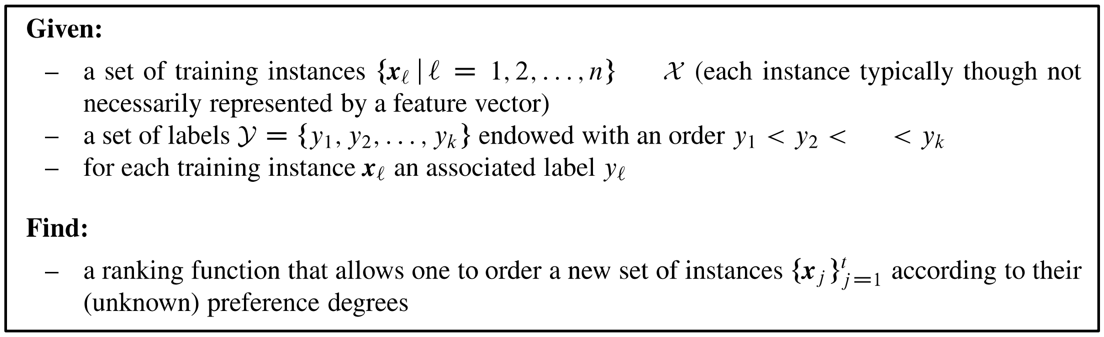
Figure 3: Label Ranking Problem [Fürnkranz11]

Boutilier et al. used Conditional Preference Networks (CP-nets), which is a qualitative graphical representation capturing preferences with conditional dependence and independence, for the label ranking problem [Boutilier04]. They explained in their paper “CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements" that every statement in the well-known CP-nets approach is formally equivalent to a label ranking [Boutilier04]. Also, many other research papers such as [Dekel04, Fürnkranz03, Har-Peled02] noticed that conventional learning problems such as classification and multiclass classification could be solved or formalized using label preferences in a label ranking problem. Har-Peled formulated and explained this in the paper titled "Constraint classification: A new approach to multiclass classification" [Har-Peled02]. For example, classification assigns a single class label to each example. Classification implicitly creates preferences among the set of labels with top 1 label is assigned. Also, multi-label classification assigns a sub-set of possible labels to each example. Multi-label classification implicitly creates preferences among the set of labels with top k labels are assigned. In both scenarios, a ranking model is needed and learned from a subset of all available preferences data in pairwise format.

Finally, there are different approaches to measure the performance of a label ranker. Normally, a loss function on rankings is used to report the predictive performance of the ranker. The loss function can be any correlation or distance measure on rankings or permutations such as   the number of incorrectly ordered pairs of labels. Some researchers improved predictions and minimized a ranking loss function's value by using a semi-supervised learning technique that reduces the disagreement of several ranking functions. There were some efforts in tackling label ranking learning problems using decision-tree learning algorithms such as CART. For example, some authors explained modifying CART to solve label ranking problems by extending the purity concept to label ranking data and learning by pairwise comparison [Fürnkranz11].

2.2.1.2 Instance Ranking

This problem is like the previous label ranking problem but the labels or classes themselves exhibit a natural order. For example, the papers submitted to a conference could be labeled as accept, weak accept, weak reject, and reject. These labels represent categories with 'accept' is ranked higher or is more preferred than weak accept and so on. Normally, training data consists of labeled instances and these labels represent preferential ranks for these instances. The goal is not to learn a classifier like in classification, but a ranking function to rank instances into ordered classes. It could be multi-classification in terms of assigning the instances to classes but these classes represent order or preferences too. Given training data composed of some instances as an input, the output would be a function that ranks and possibly assigns score to instances. Voting rules could be used in such problem and they will be discussed further more. Instance ranking problem is the same as the multipartite or k-partite ranking problem with special case of k=2 is known as bipartite ranking problem. An example of instance ranking problem would be ranking conference papers per quality by the conference chair who maybe assigns scores. The goal of instance ranking which is the proposed term by Fürnkranz is to rank instances where higher classes instances are more preferred than those from lower classes [Fürnkranz11]. Fürnkranz formalized this task as below:



Given:

- a set of training instances $\{\boldsymbol{x}_\ell \mid \ell = 1, 2, \ldots, n\}$ $\mathcal{X}$ (each instance typically though not necessarily represented by a feature vector)
- a set of labels $\mathcal{Y} = \{y_1, y_2, \ldots, y_k\}$ endowed with an order $y_1 < y_2 < \quad < y_k$
- for each training instance $\boldsymbol{x}_\ell$ an associated label y_ℓ

Find:

- a ranking function that allows one to order a new set of instances $\{\boldsymbol{x}_j\}_{j=1}^{t}$ according to their (unknown) preference degrees

Figure 4: Instance Ranking Problem [Fürnkranz11]

There are different accuracy measures for predictions in this type of problem. One example is the number of ranking errors which counts the pairs that got ranked wrong by predicting a lower class for one instance over the other while the opposite is correct. Decision-tree learning and rule learning algorithms have been used for learning such rankings and literature examined aggregating estimates into a single probability estimate. One important distinction or uniqueness to this type of problem is that instances are not in feature vector representation so a total order based on instances is needed such which isn't based on attributes rather on instances themselves [Fürnkranz11].

2.2.1.3 Object Ranking

The last preference learning task in object ranking which doesn't output any class labels to objects or instances. The goal of the object ranking problem is to learn a ranking function that receive a training data as input and outputs a ranking of these objects. It is important to notice that training data will be in the form of object x is preferred to object y like x > y. Also, such training data would be only a subset of all possible comparisons.

If we know all possible comparisons in advance, then there would be no need for prediction and putting these objects in total order would be accomplished by normal sorting algorithms such as bubble sort or quick sort. However, new algorithms are needed to find a model out of many possible models to fit training data because training data is only a subset of all possible comparisons. In literature, some approaches used by typically assigning a score to each instance and then sorting by scores. One way is to use voting rules to assign scores which will be discussed in details later. Other approaches use certain models such as trees such as LP-trees where LP means lexicographic preferences or graphs such as CP-nets where CP stands for conditional preferences. There are many types for LP-trees and one that concerns this research most is UIUP model which stands for unconditional importance and unconditional preferences.
LP-trees will be discussed later in this document.

In object ranking problems, objects are commonly represented in terms of an attribute-value representation. Training data typically represents exemplary rankings in the form of pairwise preferences in the form x > y suggesting that x should be ranked higher than y. For example, [Joachims02, Radlinski05] discussed the learning problem to rank query results of a search engine as object ranking. They explained training information can be implicitly elicited when the user clicks on some certain links in the query result than others which can be turned into binary preferences in the form selected pages are preferred over nearby pages. This object ranking problem is also known as "learning to order things" and it is summarized as:

Given:

- a (potentially infinite) reference set of objects $\mathcal{Z}$ (each object typically though not necessarily represented by a feature vector)
- a finite set of pairwise preferences $\boldsymbol{x}_i \quad \boldsymbol{x}_j$, $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{Z} \quad \mathcal{Z}$

Find:

- a ranking function $f(\,)$ that assumes as input a set of objects and returns a permutation (ranking) of this set

Figure 5: Object Ranking Problem [Fürnkranz11]

Many learning approaches for object ranking tasks exist such as dimensionality reduction methods that try to retain the preference information while reducing the data dimension. Such methods' goal is to predict a total ordering of the full set of objects given supervised total orders for certain subsets of objects. This thesis fall under this category with an extra assumption that training data objects are from a combinatorial domain and that preferences form consistent tree model. While, the number of items to be ordered in object ranking is much larger, the performance measure can be a distance function or the number of ranking errors [Fürnkranz11].

2.2.2 Techniques and Approaches for Preference Learning

Different preference learning approaches proposed in the literature that solve the three core learning tasks discussed before. Each learning task can be tackled by similar basic techniques that would be discussed in the next paragraphs. In literature, two approaches got proposed as general to preference learning which are evaluating instances based on a utility function and learning binary preference predicate after comparing pairs of instances. An important note is that researchers in literatures normally set sufficiently restrictive model assumptions about the preference relation structure. In this thesis, a tree structure is assumed and the training data would be used for identifying this structure.

2.2.2.1 Learning Utility Function

A well-established approach for modeling preferences is the approach of learning a utility function that gives a utility score to each alternative. The utility scale can be numeric or ordinal and this is discussed as regression learning or ordered classification in the literature of machine learning. In the case of label ranking problem, the utility function is learned for each label. In the instance and object ranking problems, the utility function becomes a mapping from a utility degree to each instance or object which can produce a total order. When learning a utility function, it is important to modify the conventional learning algorithms since the goal becomes maximizing ranking performance and not classification accuracy and the learner challenge becomes finding a function that agrees with as much preference data as possible. For example, [Tesauro89] was the first to formalize comparison training which is the alternative name used for object ranking. In [Tesauro89], a symmetric neural network architecture is trained with two states and a training signal for preferable state. Replacing the network's two symmetric components with a single network's state would allow the utility function to provide a real-valued evaluation. Learning the utility function for label ranking has been approached algorithmically as optimization problem by iteratively minimizing a loss function based on a least-squares approximation of the ranking error. One disadvantage of this approach is that models can not be easily explained to people. However, decision trees and LP-trees are much easier to explain which gives insight to decision-makers about the decision process which is favored in some domains like healthcare [Tesauro89, Fürnkranz11, Dunn18].

2.2.2.2 Binary Preference Relation Models

The second approach's idea is to learn a model in the form of binary preference relation which says if one alternative is better than another without using a utility function. Converting such binary preference relations into a ranking is an optimization problem because the goal is finding a ranking maximally consistent with the pairwise preferences. Normally, the objective of minimizing the number of pairs' ranks in conflict with their pairwise preference is NP-hard problem. There exist efficient techniques to deliver good provable approximations including simple voting such as Borda count of social choice theory. More complex preference structures can be derived from the preference relation to provide weak relaxed orders instead of strict linear orders. For example, Ukkonen proposed a method of leaning a bucket order which is a linear order with ties to allow indifference between two alternatives [Ukkonen09]. For label ranking problems, one predicate is learned for each pair of labels and an instance then a label ranking is calculated by weighted voting such as Borda counting using the previous pairwise predicate preferences. Different pairwise learning techniques got proposed for instance ranking problem; For object ranking problem, the relational approach has been further explored by learning a binary preference predicate then a ranking maximally consistent with these predictions generates the final ordering [Cohen99, Coppersmith06, Ukkonen09, Fürnkranz11].

2.2.2.3 Structure-based Models

The third approach is the model-based approach in which preference learning starts from specific model assumptions with an assumption such as the structure of the preference relations. Such assumption is an inductive bias that restricts the hypothesis space. If such bias is correct for the problem at hand, then the advantage would be a simplified learning problem. One disadvantage of this approach is that it is less generic since it strongly depends on the assumptions made. For example, an assumption can be that the target ranking of a set of objects is representable as a lexicographic order when these objects are represented in terms of multiple attributes. Some researchers in literature addressed learning of lexicographic orders for the object ranking problem using different algorithms. For example, a complete ranking of all objects is uniquely identified by a total order of the attributes plus a total order of each of the attribute's values. Suppose objects are made of four binary attributes then, there are 2^4 = 16 objects and 16! = 2*(10^13) rankings in total or around 2e13 which is huge. However, only 24 * 4! = 384 of these rankings can be expressed in terms of a lexicographic order. Such model in literature is called UIUP model type or unconditional importance unconditional preference tree model as discussed in [Booth10, Liu15]. Sometimes, preferences on an attribute's values depend on another attribute's values. In this case, some models like CP-nets were discussed in literature like in [Boutilier04]. CP-nets or Conditional Preference

Networks provide a graphical representation for modeling dependencies when expressing preferences among a single attribute's values like Bayesian networks. The CP-net is analogous to Bayesian networks that use conditional independence among random variables to reduce complexity of probability models. CP-net represents (in)dependencies among attributes drawn as nodes while assigning a preference relation over an attribute's values for each combination of the parent attributes' values. Some papers in literature discussed CP-networks' learning algorithms in a passive and an active setting. This CP-net model is also a quite restrictive model assumption same as the case of lexicographic orders models [Booth10, Liu15, Fürnkranz11, Boutilier04].

2.2.2.4 Aggregation and Estimation

The forth approach is local aggregation of preferences which is analogous to the idea of local estimation techniques such as the nearest neighbor estimation principle for example. In nearest neighbor estimation, a problem ranking is predicted by estimation based on "neighbored" rankings observed in input dataset then obtaining a final ranking using averaging-like aggregation operator on these "neighbored" rankings. While this approach is flexible and doesn't impose a specific model assumption, it implicitly assumes consistency underlying the nearest neighbor inference principle. For label ranking problem, the nearest neighbor approach gets the query's k nearest neighbors then applies any aggregation technique like voting theory or the average to combine the different rankings into a prediction. For other types of preference learning problems like object ranking, aggregation techniques got used also to combine separate rankings into a complete ranking of all objects. One practical application is information retrieval when different rankings of different search engines would be combined into an overall ranking. Another example is ranking sports players in a certain game when judges have different rankings that must be aggregated into an overall competition ranking. Voting theory play a vital role in such scenarios and different voting rules have been proposed that will be examined further in this thesis [Fürnkranz11].

2.2.3 Preference Learning Applications

Preference learning problems and ranking problems appear naturally in various domains. In the previous section, the different preference learning problems got discussed and categorized per the learning task (label, instance, or object ranking) [Fürnkranz11]. Also, different approaches or learning technique found in literature got explained such as (learning utility functions, learning binary preference relations, learning preference models having a specific structure, or using local estimation and preference aggregating methods) [Fürnkranz11]. It is important to notice that any combination of a task and a technique from above can be found in literature. Sometimes one research paper can be about several categories since learning a utility function or a binary preference relation commonly used with other techniques. Preference learning is important for many application areas such as recommender systems and search engines. For example, search results can be ranked per a user's preferences and new product recommendations such as cars can be ranked per a customer's preferences based on the features of different car models. When ranking search results, an unknown preference relation would be learned from user feedback implicitly collected via their clicking behavior on past queries' results rankings. Many research on information retrieval literature used LETOR or 'LEarning TO Rank' package that contains queries with user feedback datasets [Fürnkranz11]. Preference learning is also important for recommender systems which online stores use to recommend products to customers. There are other approaches for recommender systems like collaborative filtering systems that provide recommendations based on user similarity, and other systems that provide recommendations based on item similarities. Some preference learning approaches learn decision trees or models to recognize recommendation features to predict preference ranking [Fürnkranz11].

2.3 Genetic Algorithms

The idea of the genetic algorithm is inspired by the natural selection theory in biology. In that theory, natural selection ensures the population's survival for the fittest [Dewdney93]. The same idea is applicable to the genetic algorithm where it will move from among generations of possible solutions. 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. However, the steady state does not always mean that no improvement can be done since there is a chance that the fitness measure get stuck for long time at a local optimum. Assume a function F is used as a fitness measure for a problem. If F has many local maxima, then the genetic algorithm will run longer to find the true maximum.

Early computer science pioneers were interested in biology and psychology beside computers too and were inspired by natural systems. From the earliest days, computers were used to model the human brain, mimic human learning, and simulate evolution in biology. Since 1980s, biologically motivated computer research grown into separate fields such as neural networks, machine learning, evolutionary computation that includes genetic algorithms [Mitchell99].

Many researchers started to examine evolutionary systems in 1950s as inspiration to optimize engineering problems. The idea was to develop a population of candidate solutions to a certain problem using operations such as natural selection and variation. Evolutionary computation consists of evolution strategies, evolutionary programming, and genetic algorithms [Mitchell99]. Later, many algorithms were inspired by evolution and developed for optimization and machine learning. John Holland invented and developed Genetic algorithms (GAs) in the 1960s at the University of Michigan [Mitchell99]. Hollands paved the theoretical foundation for adaptation using genetic algorithms by publishing his book: Adaptation in Natural and Artificial Systems. Holland's method was moving from one population of solutions to another using selection and genetic-inspired operators such as crossover, mutation, and inversion. Solutions were called chromosomes and each chromosome was encoded as a string. Each chromosome composed of many traits, features, attributes, or simply "genes" with each gene being one value or "allele". The selection process picks the fittest chromosomes in the populations to reproduce children. This is proportional to one's fitness so the fittest chromosomes will produce more children on average. The crossover process will recombine subparts of two chromosomes to produce a new chromosome. Mutation process, which mimic genetic error while copying, will modify the allele value of any location in the chromosome randomly. Inversion process simply reverses the order of values in a section in the chromosome [Mitchell99].

The evolution mechanisms are suited for some computational problems that require searching a massive number of candidate solutions. Protein engineering is an inspiring example of such problems that search among the vast number of possible sequences for a protein. This relates somehow to the preference mining problem. These search problems benefit and make use of parallelism since different solutions are explored simultaneously [Mitchell99].

Multiple processors could measure the fitness of chromosomes simultaneously providing computational parallelism plus an intelligent strategy could provide a boost like smart heuristics that help prone search space in A* algorithm. In evolutionary computation, natural selection theory is the rule that guarantee variation due to crossover or mutation; which lead to emergent behavior of designing high−quality solutions to problems with ability to adapt if environment evolves. An evolutionary-based learning algorithm could better adapt to feedback or new information because of the adaptability feature of evolutionary computation. Biological Evolution is an inspiration because it is a searching technique that look up for a solution among many possibilities. Evolution is a method of finding innovative solutions to complex problems. Sometimes, the fitness function is not fixed and can be continually changing as population evolve, so evolution will be searching a constantly changing candidate solutions. This can happen in preference mining problem when you learn a new preference relation and add it to the old preferential data or old requirements. Searching for solutions with changing conditions is requires adaptive programs. Therefore, evolution is a huge parallel search program rather than exploring one solution or path at a time. Evolution evaluate and changes millions of solutions in parallel [Mitchell99]. At the end, evolution rules are not difficult and the population evolve by random variation via crossover, mutation, and other operators; then natural selection guarantees the fittest to survive and reproduce propagating their fit or good genetic features to future generations.

Searching in a "search space" is a famous problem type in computer science in which a computer tries to find a goal solution among a huge collection of candidate solutions. The term "search space" means the collection of candidate solutions to a problem and there is sometimes some notion of distance between them. Genetic algorithms assume that good *parent* candidate solutions in the space can be combined by crossover produce high−quality *offspring* candidate solutions. Fitness values can form a landscape with hills, peaks, or valleys like physical landscapes. Under Wright's formulation, evolution move populations along landscapes, and "adaptation" guide the movement toward local peaks or local optimum. Crossover and mutation in genetic algorithms lead population while moving around on the fitness function landscape. It important to say an individual fitness must be relative to current population. Candidate solutions are assigned a fitness value relative the other solutions in the population [Mitchell99].

Chapter 3

PREFERENCE VISUALIZATION

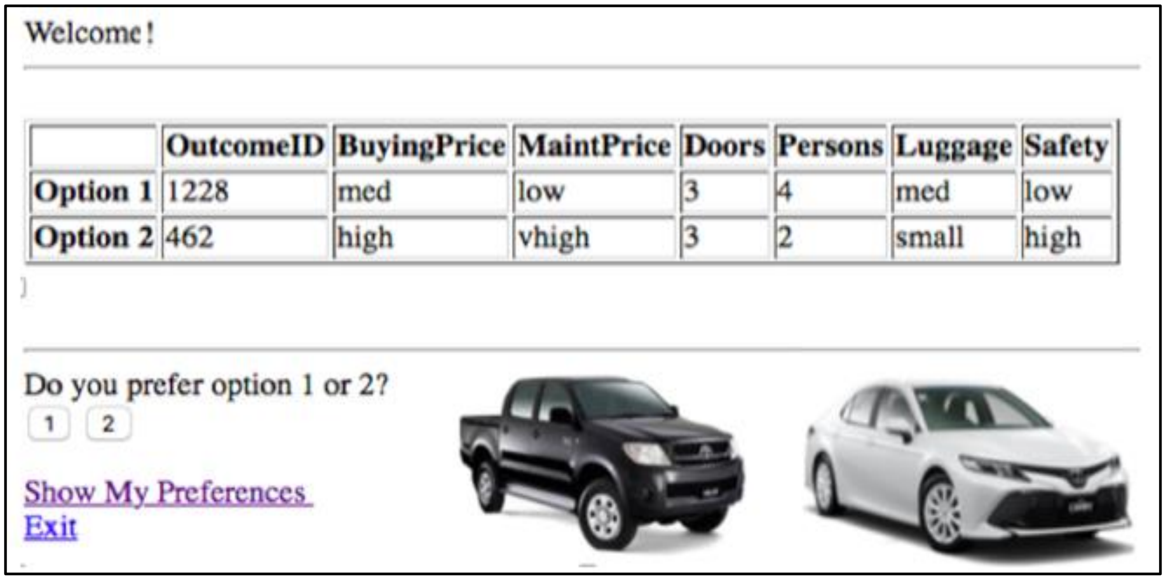
A framework was developed for preference visualization. The functional product could be tested in real world in the future to understand and quantify its potential impact. This framework is helpful and applies my new technique to let people understand their own preferences. This framework is useful to industry and companies in predicting their customers' preferences and purchasing behavior. This chapter outlines the design aspects of the visualization framework. The framework is web-based with the objective of learning user preference models (e.g., LP-trees and CP-nets) through interacting with the user. Python and Django version 1.11 were used for developing the framework.

The web-based framework visualizes various types of LP-Tree models (UIUP, UICP, CICP) learned by an existing greedy learning algorithm developed by Liu in C++ [Liu15]. The system consists of three parts or phases to: (1) elicit user preferences, (2) learn graphical preference models (UIUP, UICP, CICP), and (3) visualize these models to the user in different representations (Outline, GUI, JSON, XML). Different modules are built for the elicitation of user preferences and for visualization of learnt models. The existing greedy learning algorithm. In summary, the framework gives these options to the users:

1. Play games as a way for preference elicitation to generate a training dataset
2. Learn different types of trees (UIUP, UICP, CICP)
3. Visualize the different trees in different formats (Outline, GUI, JSON, XML). D3 library is used to create the GUI representation.

3.1 Elicitation Phase

Preference elicitation is done via a game playing approach where the user can play as many games as possible. On each game, the framework shows two options and the user must pick the one s/he prefers most. Text is currently used to describe the two options like in Figure 6:



Welcome!

	OutcomeID	BuyingPrice	MaintPrice	Doors	Persons	Luggage	Safety
Option 1	1228	med	low	3	4	med	low
Option 2	462	high	vhigh	3	2	small	high

Do you prefer option 1 or 2?

Figure 6: Page to capture user preferences via playing games

The framework may use images beside text to show the two products as above. A lot of dataset in different domains are gathered. Datasets with images (car domain, watches domain, houses) got collected too. The framework typically use one dataset of products. In each game, it picks two products and show them to the user. Then the user must pick the one that he/she prefers most. Text description of outcomes is only implemented like in the figure 6. First, the user logins to the framework and then plays as many games as possible. On each game, the framework shows the user two options and the user must pick one. If a user picks option one over option two, then this implies the user prefers option 1 over two. Each option has different values for each attribute. As an example, the above image show 6 attributes in the car domain which are “Buying Price”, “Maintenance Price”, “Number of Doors”, “Number of Persons”, “Luggage Size”, and “Safety Level”. This is a combinatorial domain and each attribute has different values that can be combined to reach different outcomes. A random combination is used to reach a random option. However, there was a need for real-life datasets to make this framework useful for real-life applications. Many different datasets were gathered including images that represent the outcome from different domains such as Cars domains, Houses domain, Watches domain, and People Talents domain. Each domain has its own attribute names and each attribute has its values. It is a good idea to use pictures and ask the user to compare two pictures. This is a better approach since each picture visualizes the attribute values. For example, if a car has an attribute called Color and the value is red in outcome 1 and black in outcome 2 then the two pictures will show such contrast easily. Showing pictures combined with textual descriptions is planned since users generally prefer pictures over text. Below is an overview of the architecture diagram of the framework:

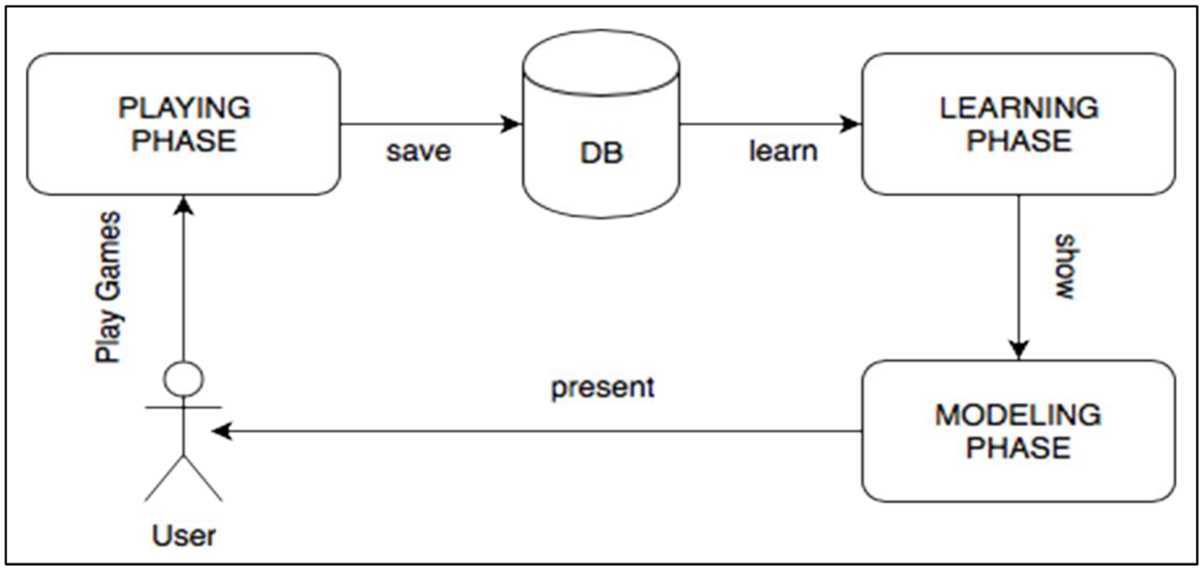
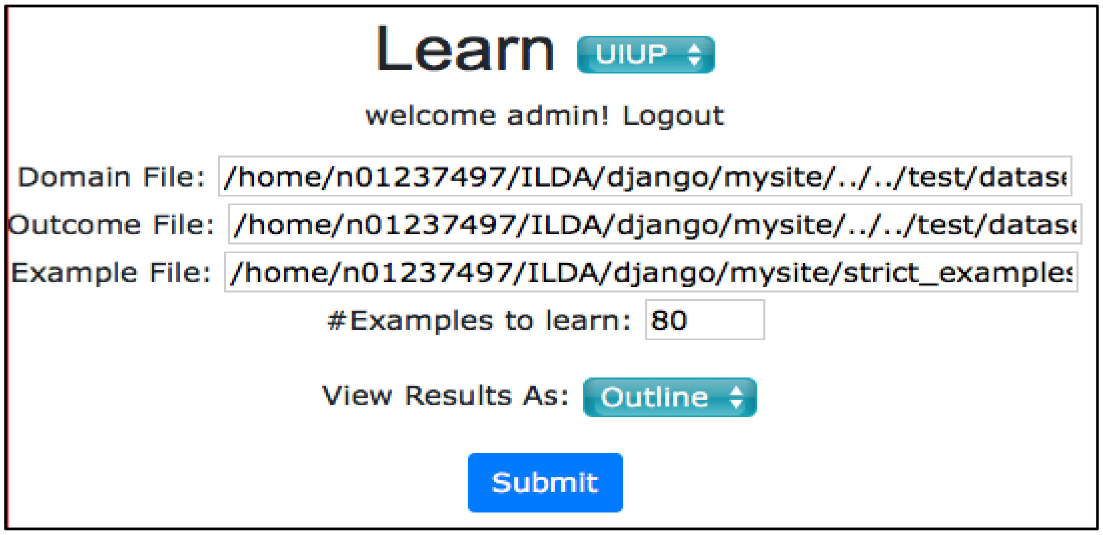


Figure 7: Architecture Diagram

3.2 Learning Phase

In this phase, the framework provides the user with the option to pick one LP-Tree model to learn and how to present this learnt model like in Figure 8. Then, the framework would send the set of examples gathered during the elicitation phase to the greedy algorithm as the training dataset along with domain description files. The greedy algorithm learns the selected tree model like UIUP from the dataset and returns a pointer to a tree object in C++. Hence, more development needed and implemented during this thesis to bring this C++ tree object from a computer memory to a usable visualized model. The following sections discuss the details of such development's implementation.


Learn

welcome admin! Logout

Domain File: /home/n01237497/ILDA/django/mysite/../../test/datas

Outcome File: /home/n01237497/ILDA/django/mysite/../../test/datas

Example File: /home/n01237497/ILDA/django/mysite/strict_example

#Examples to learn: 80

View Results As: Outline

Figure 8: Page to learn a certain type of preference tree

Django framework was used to create a poll app that ask users for polls. These polls are a way to capture user preferences directly. Django is integrated with the C++ preference learning libraries developed using pybind11 [Liu15]. A web form was created to read in three files (Domain Descriptor File, Outcomes File, Examples File) and the number of examples to learn. Then, these data were used for learning a preference model.

A function named treeToString was developed to convert or serialize a tree structure to a string representation. This can be helpful in encoding trees into chromosomes for genetics algorithms. A tree traversal function was implemented in python to be used directly in Django Web App. JS library called D3 was used to represent the C++ preference tree structure into a graphical tree that the user can interact with by expanding nodes and collapsing nodes.

A function was implemented to convert the preference tree structure into XML format. A Drop Menu control is added that allows the user to select the way to present the learned preference model. Some of these visualization options are Outline, XML, JSON, GUI. The GUI improves understandability and readability by adding labels to the edges, by appending local preference information to nodes, by using colors to mark parents through a path, and by using greater than symbol among local preferences.

A page was developed to capture the user input over two options and store them in a database for later usage by the learning algorithm. Support was added to save the learned model into a JSON file format. Then, a web page was created where the user can play games where he picks his preference among two options. The Car Domain was used to generate two car objects with random attributes and value pairs presented in a table. Then, the game results were saved into the database as one example with the two options and the user preference.

The framework was enhanced by adding an authentication system to allow multiple users use the framework. The users can independently report their preferences. This was done by enforcing a login page to authorize the user before playing the games unlimited number of times until the user logs out.

The database was used to dynamically generate a strict_examples file for each user then fed these data to the algorithm to learn a model.

Amazon Mechanical Turk was used to create surveys that integrate with the web framework. Also, one example of hits on Amazon Turks was implemented to compare two images with the attached image descriptions. The amazon survey results were exported into a key-value pairs.

3.3 Modeling Phase

In this phase, the framework presented a certain model to the user. There are different visualization formats such as: Collapsible List/Accordion, Graph, GUI (using D3 lib), JSON, and XML. Visualization support were added to all the different tree types such as UIUP, UICP, and CICP. While, JSON and XML representation formats are well known, Figure 9 shows the other different visualization formats of different models; UIUP model in collapsible list format is shown on the left, UICP model in graph format is in the middle, and CICP model in GUI format on the right.

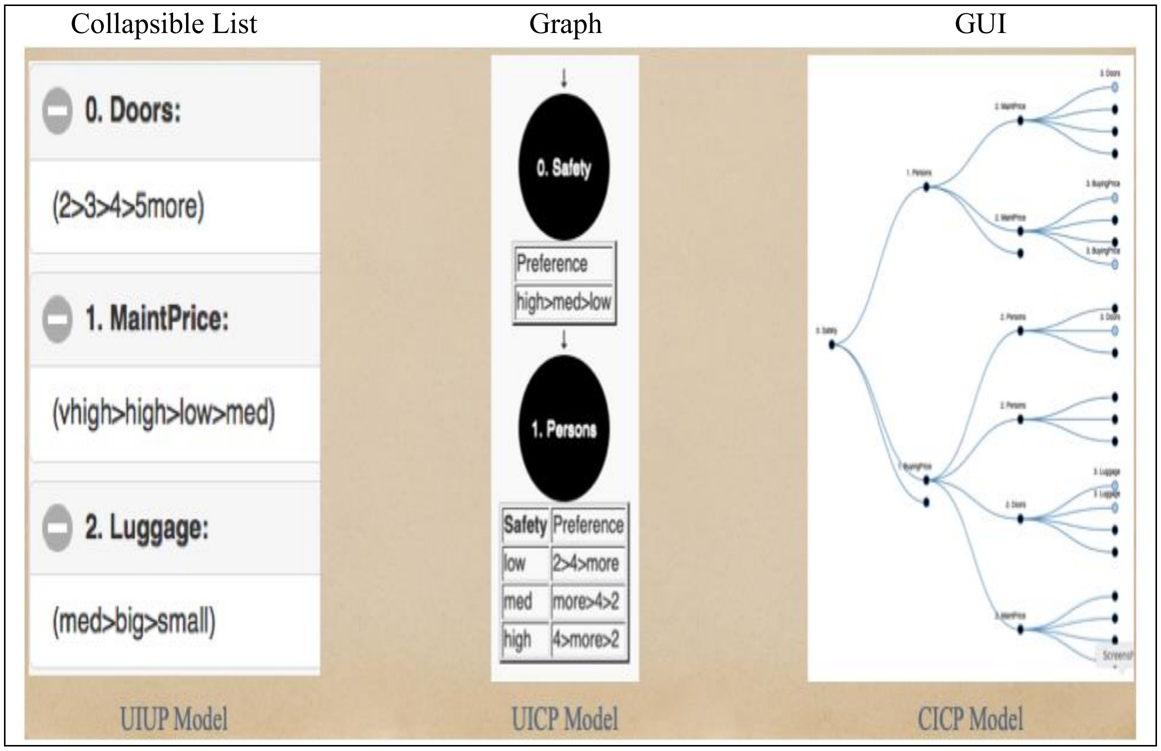


Figure 9: Different Visualizations of Different Models

The framework was deployed on a Linux server using Apache and WSGI. The use of graphs improved understanding of the UIUP and UICP models since big circles represented attributes order and tables used to describe conditions and/or preferences as in Figure 10 and Figure 11.

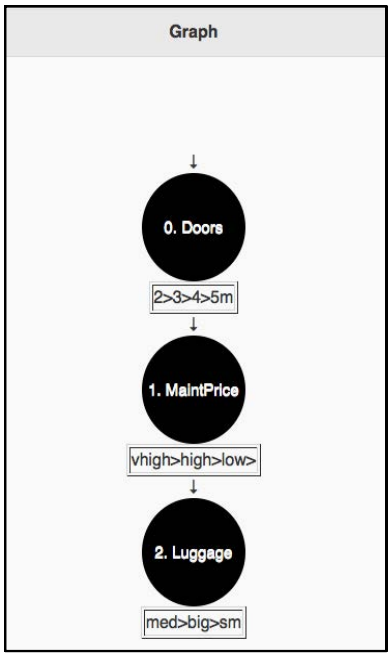
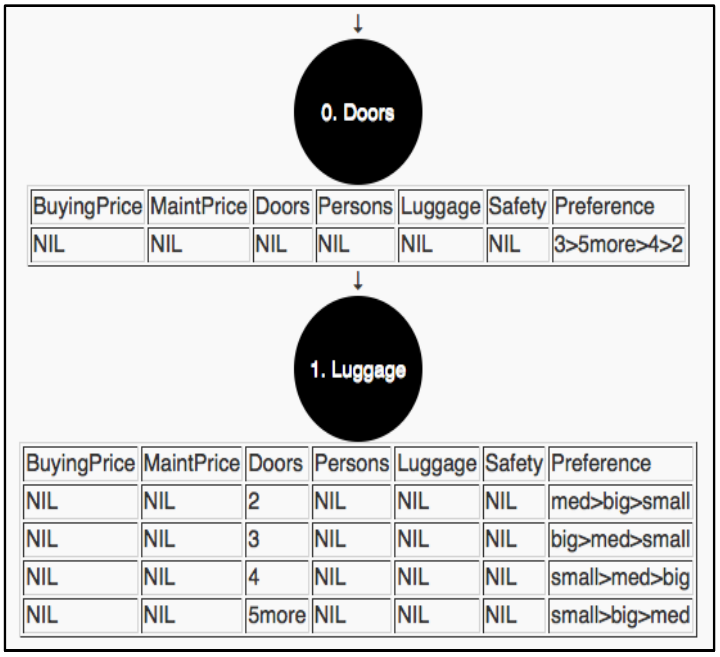


Figure 10: UIUP Using Graph



BuyingPrice	MaintPrice	Doors	Persons	Luggage	Safety	Preference
NIL	NIL	NIL	NIL	NIL	NIL	3>5more>4>2

BuyingPrice	MaintPrice	Doors	Persons	Luggage	Safety	Preference
NIL	NIL	2	NIL	NIL	NIL	med>big>small
NIL	NIL	3	NIL	NIL	NIL	big>med>small
NIL	NIL	4	NIL	NIL	NIL	small>med>big
NIL	NIL	5more	NIL	NIL	NIL	small>big>med

Figure 11: UICP Using Graph Presentation

The framework was improved to allow the user easily switch among different tree types (UIUP, UICP, CICP) and different visualizations formats like Outline, Accordion, GUI, Graph, JSON, XML). D3 library was used to visualize the trees in the GUI format as shown with CICP model in Figure 9.

It is important to notice that LP-Trees are different than other trees in two aspects. The first one is that the order of nodes in the tree from the root to the leaf has a useful meaning which is the importance of the attribute. The second one is that the order of outgoing edges from a certain node to all its children matters and has another meaning which is local preferences for values of this specific attribute with first left most edge is the most preferred.

Chapter 4

PREFERENCE LEARNING

4.1 Genetic Algorithm (GA)

A genetic algorithm was implemented to learn UIUP models from preferential data. The genetic algorithm performs the following steps while repeating last three steps repeatedly for 100 generations:

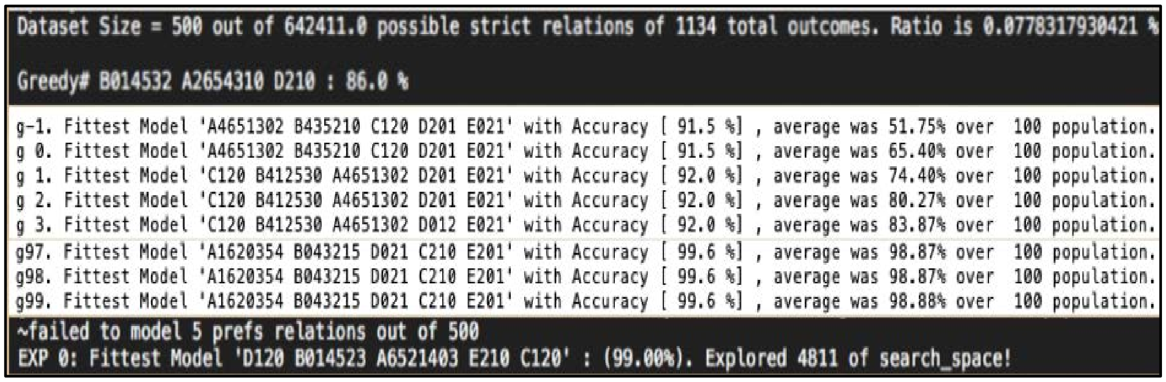
0. *Initialization*: randomly create 100 chromosomes like (B201 A210 C012). These 100 chromosomes are called the *seed*.
1. *Fitness Function*: evaluate each chromosome based on the number of examples (#examples) it can correctly classify. This is called the *accuracy* of a chromosome.
2. *Selection*: select the best/top 100 chromosomes overall as a current population. This technique is called elitism selection as explained in chapter 2. These 100 chromosomes are referred to as the *elite* in this thesis.
3. *Reproduction*: produce new variants from the elite. These variants are called *children*. This is done via crossover and mutation operations that are the source of exploitation and exploration of the search space. The mutation operation shuffles

the order of attributes and values of one randomly selected attribute. The crossover operation keeps the longest common prefix (agreement) between two parent chromosomes then shuffle the remaining; one of the two parent chromosomes is the best chromosome found over all generations.

Attribute	**Values**
BuyingPrice:	vhigh, high, med, low
MaintPrice:	vhigh, high, med, low
Doors:	2, 3, 4, 5more
Persons:	2, 4, more
Luggage:	small, med, big
Safety:	low, med, high

Table 2: The Cars Domain Used

Extensive experimentations were performed on the genetic algorithm to test out different implementations and techniques for selection, mutation, crossover, population size, and number of generations. The genetic algorithm showed promising results, but it was trapped in local optima in some cases. The genetic algorithm was compared against the greedy implementation. The genetic algorithm outperformed the greedy approach in accuracy. However, when the training dataset size was huge, the genetic algorithms started to take longer time to evaluate candidate solutions. Cars Domain is used as shown in Table 2. Figure 12 shows the output of the genetic algorithm program during the learning process. The program prints some information about the input dataset and the accuracy of the model learned by the greedy algorithms. The program starts performing the genetic algorithm steps while reporting best accuracy found and the average accuracy of the population. At the end, the program reports the fittest model and it accuracy.



```
Dataset Size = 500 out of 642411.0 possible strict relations of 1134 total outcomes. Ratio is 0.0778317930421 %

Greedy# B014532 A2654310 D210 : 86.0 %
g-1. Fittest Model 'A4651302 B435210 C120 D201 E021' with Accuracy [ 91.5 %] , average was 51.75% over  100 population.
g 0. Fittest Model 'A4651302 B435210 C120 D201 E021' with Accuracy [ 91.5 %] , average was 65.40% over  100 population.
g 1. Fittest Model 'C120 B412530 A4651302 D201 E021' with Accuracy [ 92.0 %] , average was 74.40% over  100 population.
g 2. Fittest Model 'C120 B412530 A4651302 D201 E021' with Accuracy [ 92.0 %] , average was 80.27% over  100 population.
g 3. Fittest Model 'C120 B412530 A4651302 D012 E021' with Accuracy [ 92.0 %] , average was 83.87% over  100 population.
g97. Fittest Model 'A1620354 B043215 D021 C210 E201' with Accuracy [ 99.6 %] , average was 98.87% over  100 population.
g98. Fittest Model 'A1620354 B043215 D021 C210 E201' with Accuracy [ 99.6 %] , average was 98.87% over  100 population.
g99. Fittest Model 'A1620354 B043215 D021 C210 E201' with Accuracy [ 99.6 %] , average was 98.88% over  100 population.
~failed to model 5 prefs relations out of 500
EXP 0: Fittest Model 'D120 B014523 A6521403 E210 C120' : (99.00%). Explored 4811 of search_space!
```

Figure 12: GA Program

Many experiments were conducted using the cars domain in Table 2 to compare the genetic algorithm against the greedy algorithm in terms of accuracy (Figure 13) and execution time (Figure 14). Different training dataset sizes were used from 100 to 1000 examples. Dataset sizes are plotted on the x-axis below in Figure 13 and Figure 14. Each experiment was run for ten times per each dataset size and the average accuracy and execution time of both the genetic algorithm and the greedy algorithm were recorded. Figure 13 is a plot comparing the accuracy of the genetic algorithm vs the greedy algorithm. Figure 14 is a plot comparing the time needed in seconds to learn a UIUP model using the genetic algorithm and the greedy algorithm. There is a clear tradeoff between accuracy and time; the genetic algorithm is more accurate while the greedy algorithm is faster.

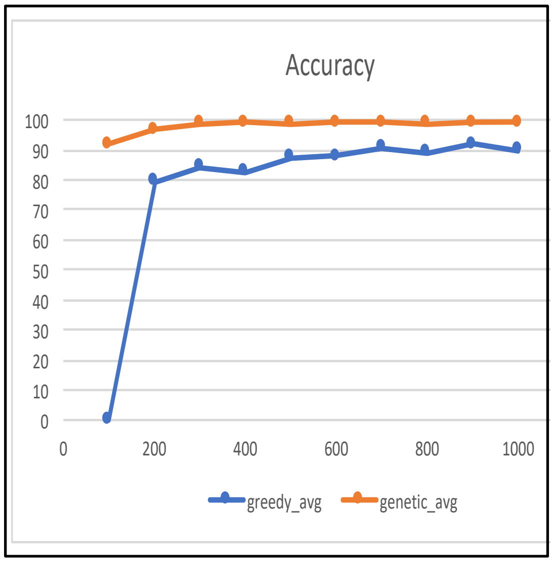


Figure 13: Accuracy

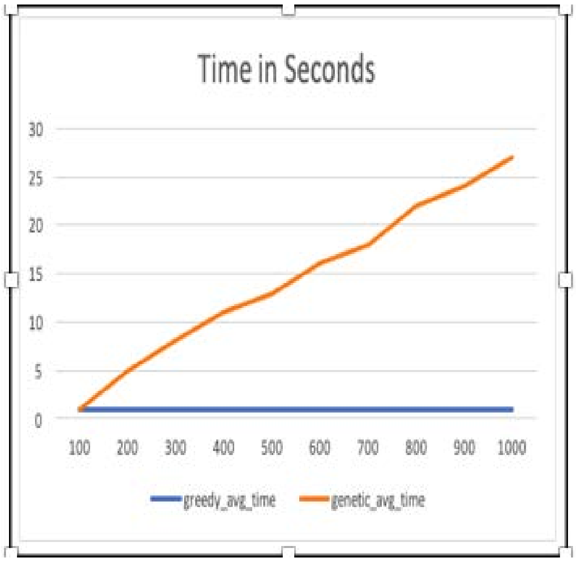
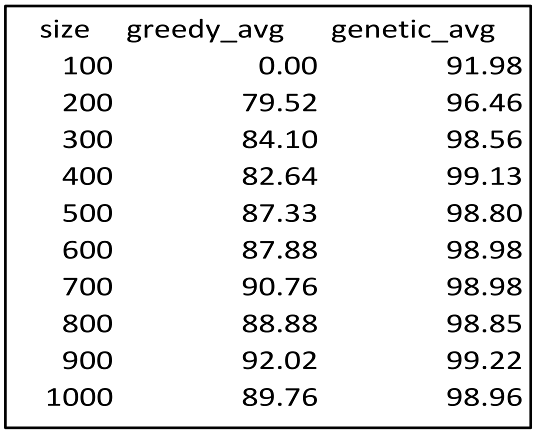


Figure 14: Time in seconds

After testing both the accuracy and the time using the input training dataset, further testing was done to check the prediction performance on unseen dataset for the models generated from both algorithms. The unseen dataset size contained 10k examples. Figure 15 and Figure 16 show the prediction accuracy of both the genetic and greedy algorithms. The genetic algorithm was around 10% more accurate on average than the greedy algorithm.



size	greedy_avg	genetic_avg
100	0.00	91.98
200	79.52	96.46
300	84.10	98.56
400	82.64	99.13
500	87.33	98.80
600	87.88	98.98
700	90.76	98.98
800	88.88	98.85
900	92.02	99.22
1000	89.76	98.96

Figure 15: Prediction Accuracy Scores

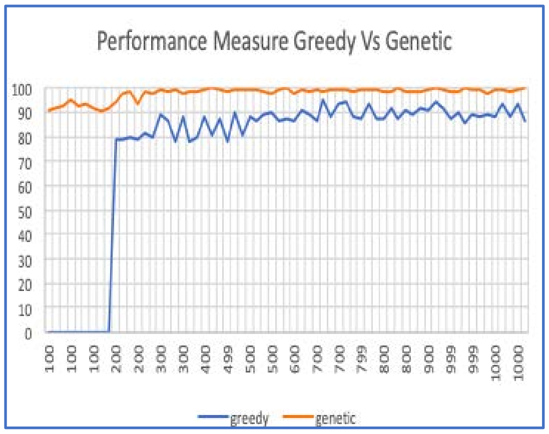


Figure 16: Testing Accuracy on unseen dataset

Using the voting theory of the computational social choice field was tested. The program implemented many voting rules to make two parent reproduce better children or to reach to an ordering based on a voting profile of two solutions. Such fast algorithms based on voting do not necessarily yield an acceptable agreement. To conclude, voting-based heuristics did not help much in guiding the genetic algorithm into the right path of finding an optimal solution. Two main operations were performed which are mutations and crossover. These two operations were performed twice for importance of attributes and for local preference ordering of one attribute's values. These operations helped in finding optimal models because crossover always kept the better solutions in the pool of elite chromosomes leading to better chromosomes evolution across generations.

4.2 Dynamic Programming Algorithm (DPA)

An exact algorithm to learn optimal UIUP tree models from noiseless and noisy data is introduced in this thesis. The algorithm was devised after discovering a way to speed up the genetic algorithm evaluations which represented a bottleneck via transforming the initial dataset into a summary table. This phase is called data preprocessing and transforms data from one format to another. For this summary table, all information that are useless for decision making such as similar attribute's values are removed. Understanding Held–Karp algorithm is important since its idea is relatively close [Held62]. The DPA algorithm is explained below. First, assume Table 3 is the domain with attributes A, B, C:

A [PRICE]	B [COLOR]	C [MODEL]
0 (cheap)	0 (white)	0 (honda)
1 (average)	1 (black)	1 (toyota)
2 (expensive)	2 (green)	2 (nissan)

Table 3: Cars Domain

In other words, attributes are alphabets like A, B, C and values are indices like 0, 1, 2. From the above combinatorial domain, there are 27 different objects, combinations of values of the three attributes. Now we need to build a model from a training dataset to learn the user preferences or total order over these products. For example, when buying a car, you may consider the most important attribute of a car to be the price. The second most important attribute is the color. Finally, the least important property is the model. Besides having order for the importance of attributes, you also might have preferences over the values within each attribute like prefer red color over white color. This is called UIUP model. Another example, when choosing between two same-price flights, one may prefer a nonstop flight to a flight with stops. A UIUP model can capture such preferences and it can be either represented graphically or textually as demonstrated below in Figure 17.

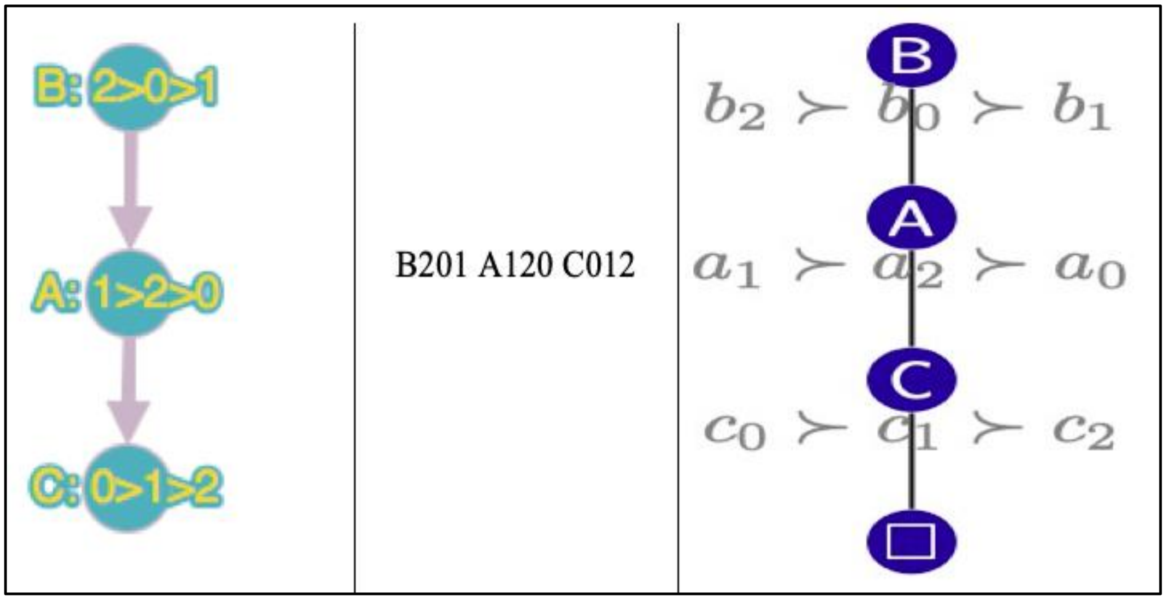

B: 2>0>1
A: 1>2>0
C: 0>1>2
B201 A120 C012
B
$b_2 \succ b_0 \succ b_1$
A
$a_1 \succ a_2 \succ a_0$
C
$c_0 \succ c_1 \succ c_2$

Figure 17: Different representations for the same UIUP Model

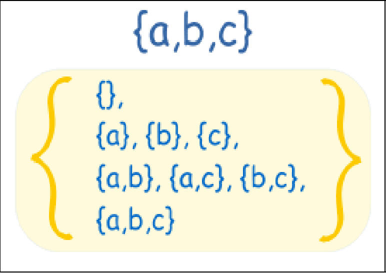
Given two products such as ([A2 B0 C0], [A0 B0 C2]) or simply ([200], [002]), the above model can predict and entail that product [A2 B0 C0] or [200] will be preferred by the user over the other one. This is because the value for attribute B is the same for both but the second attribute in importance, which is A, has value 2 in the first product which is preferred over value 0. Many studies on human decision making experimentally demonstrated that humans often make decisions using lexicographic reasoning [Yam10]. Humans tend not to use mathematically sophisticated methods like linear additive value maximization when facing or weighting alternatives [Yam10]. The next section explains how the model was built and presents the problem formalization. Table 4 contains the sample observations that will be used as the input training dataset.

Problem Statement: Given a pairwise preferential data records in the form of two products per record where the user prefers first option over the second, find or learn UIUP tree which is a Lexicographic Preference Model (LPM) that correctly classifies maximum number of records from the training dataset.

X > Y Observations	X:<ABC>	Y:<ABC>
ob1	002	202
ob2	021	201
ob3	101	020
ob4	110	002
ob5	111	120
ob6	111	221
ob7	120	212
ob8	200	000
ob9	202	002
ob10	211	122

Table 4: Observations or the input training dataset

Implementation of the Greedy-Brute-force learning algorithm for noisy data gave *Greedy-Model*: [A102 B12] with *70% accuracy* meaning it failed to predict 3 out of 10. For the DPA algorithm, the initial training dataset must be transformed into a summary table. The Summary Table's idea is simple. The dataset is processed once to place observations into buckets. Observations are divide into buckets based on the deciding attributes only. For example, if attribute A had different values for both options/products in one observation and all other attributes' values were the same then add this observation in the bucket of 100 meaning 'A' was only the deciding factor. Zero in a bit-mask means that attribute at that index was useless attribute in decision making process or in other words both products had the same exact value of that attribute. XOR-ing two outcomes gives a Mask ID. Figure 18 shows mask ID in range [0 –7] when having three attributes.



$\{$	$A,$	$B,$	C	$\}$
	1	1	1	binary
	1	2	4	$=7$

$$\begin{array}{ccccccc}
0 & = & 0 & 0 & 0 & = & \{\} \\
1 & = & 1 & 0 & 0 & = & \{A\} \\
2 & = & 0 & 1 & 0 & = & \{B\} \\
3 & = & 1 & 1 & 0 & = & \{A,B\} \\
4 & = & 0 & 0 & 1 & = & \{C\} \\
5 & = & 1 & 0 & 1 & = & \{A,C\} \\
6 & = & 0 & 1 & 1 & = & \{B,C\} \\
7 & = & 1 & 1 & 1 & = & \{A,B,C\}
\end{array}$$

Figure 18: Mask ID from the power set

Table 5 is an example for the summary table of the input training dataset. Data Pre-processing from raw data into a summary table can happen offline or on-the-fly. For example, My Learning Algorithm can read the summary table as input which would have been stored in the database as in Table 6. Parallel machines may write preferential data directly into a summary table in the database allowing for big data processing. The normalized summary table in the database will look like Figure 18 below; we have 777 observations where Attribute 'A' was the deciding factor with value 1 preferred to 2. The summary table provides a huge input reductions as one record in a summary table represent 777 observations with same A preferences; value $U = 1$ is preferred to $V = 2$.

Mask	**Observations**	A	B	C	Total
011	ob5		{12:1}	{10:1}	1
100	ob1, ob8, ob9	{20:2, 02:1}			3
101					
110	ob2, ob6	{02:1, 12:1}	{20:1, 12:1}		1
111	ob3, ob4, ob7, ob10	{10:2, 12:1, 21:1}	{02:1, 10:1, 21:1, 12:1}	{10:1,02:2,12:1}	5

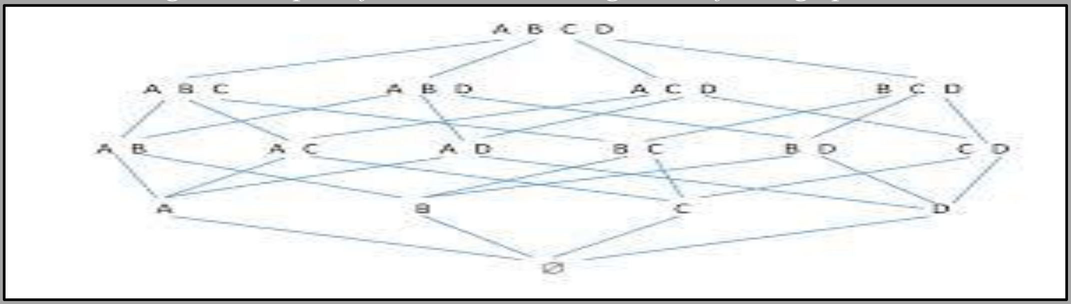
Table 5: Summary Table

MaskID	Attribute	U	V	Count
100	A	1	2	777

Table 6: Storing Summary Table in a relational database

The DPA algorithm converts the training dataset into the above summary table and produces an optimal UIUP model. It consists of two important algorithms. The core idea of summary table is to record differences only. For example 001 > 000 would be added as ([001, C, 1, 0, +1]) where 001 is the mask, C is the attribute where both products have different values, 1 & 0 are the values of the left & right product and +1 is the total count. DB Memory Complexity is: $(2^N) * N * M * M$ where N = #attribute, M = max #values. The Dynamic Programming algorithm breaks the learning into sub-problems. Each sub-problem will have optimal memorized result so avoiding computing the same results again. This complex problem has two properties which are overlapping sub-problems and optimal substructures. Check Figure 19 and algorithms 1 & 2 for clarifications.

reducing search space from N! to 2^N is huge when finding optimal order.



if N = 10 then we explore 1,024 only instead of 3,628,800 states.
if N = 20 then we explore 1,048,576 only instead of 2.4e18 states.

Figure 19: Find Optimal Order Task

A- *Algorithm 1*: Find local preferences' optimal order given a graph matrix:

Input: Matrix

Output: Optimal permutation for local preferences of graph nodes

Table 7 below shows current observations that depend on Attribute A's values.

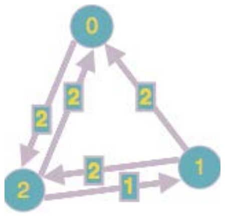
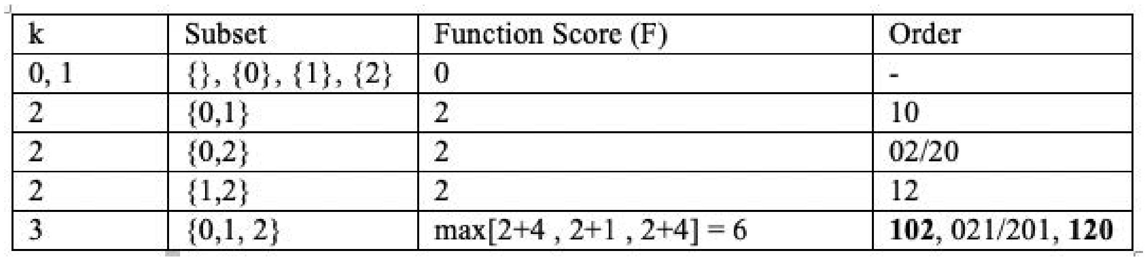
Attribute *A* can help classifying or deciding these observations	ABC Format	Total
{ob1, ob2, ob3, ob4, ob6, ob7, ob8, ob9, ob10}	002 > 202	9
	021 > 201	
[A] | ***0*** | ***1*** | ***2***	101 > 020	
0 | 0 | 0 | 2	110 > 002	
1 | 2 | 0 | 2	111 > 221	
2 | 2 | 1 | 0	120 > 212	
A's matrix	200 > 000	
	202 > 002	
A's graph	211 > 122	

Table 7: Finding optimal order for attribute A's values

Dynamic Programming Solution: Algorithm 1 of DPA has an optimization property which is every sub-solution of the optimal solution/permutation is itself optimal to the sub-problem. From the above matrix, the optimal order will be found using dynamic programming. Like solving an asymmetric Traveling Salesman Problem, the problem at hand was converted into an integer linear programming formulation. Here are the calculations in Table 8:



k	Subset	Function Score (F)	Order
0, 1	{}, {0}, {1}, {2}	0	-
2	{0,1}	2	10
2	{0,2}	2	02/20
2	{1,2}	2	12
3	{0,1, 2}	max[2+4 , 2+1 , 2+4] = 6	**102**, 021/201, **120**

Table 8: DPA Calculations like Held–Karp algorithm for Traveling Salesman Problem

Optimal order is (102) = 2+2+2 = 6 or (120) = 2+2+2 = 6 out of 9. Results are presented in Table 8. Let k be the size of the subset, so when k = 1, consider sets of one element:

F ({0}) = 0, F ({1}) = 0, F ({2}) = 0. When k = 2, consider sets of 2 elements:

F ({0, 1}) = max[r(01), r(10)] = max[0, 2] = 2 via order (10). r(10) = ray "10" score.

F ({0, 2}) = max[r(02), r(20)] = max[2, 2] = 2 via order (20) or order (02).

F ({1,2}) = max[r(12), r(21)] = max[2, 1] = 2 via order (12).

When k =3, F ({0, 1, 2}) = $max[\, F\ (\{0,1,2\} - \{i\})\ +\ g(i)\ :\ i = 0,1,2]$

Where g(i) = sum of all incoming arc weights or simply *i's* weighted incoming degree. In summary, reduce a set to subsets of size k-1 before visiting each i last. Assume x is the set of values, then here is the recursive function that the algorithm seeks to optimize:

$$f(x) = \max\left\{ f(x-i) + g(i) \right\}_{i \in x}$$

Brute-force Solution Characteristics:

1) Generates all (m)! permutations of m values.

2) Calculates cost of every permutation and store costs.

3) Returns the permutation with optimal cost.

4) Time Complexity: Θ(m!)

5) Advantage: Returns Optimal Solution.

6) Drawback: Computationally prohibitive.

7) Used in the greedy algorithm.

B- *Algorithm 2*: Learn UIUP Model:

<u>Input</u>: Examples set of pairwise comparisons and attributes set

<u>Output</u>: Optimal UIUP Model

Now, *Algorithm 2* will call *Algorithm 1* so many times with different matrices as input to optimize the overall learning problem. It is like one big Traveling Salesman Problem of so many smaller Traveling Salesman Problems. Figure 20 demonstrates the sub-problems that the dynamic programming will memorize its optimal sub-solutions.

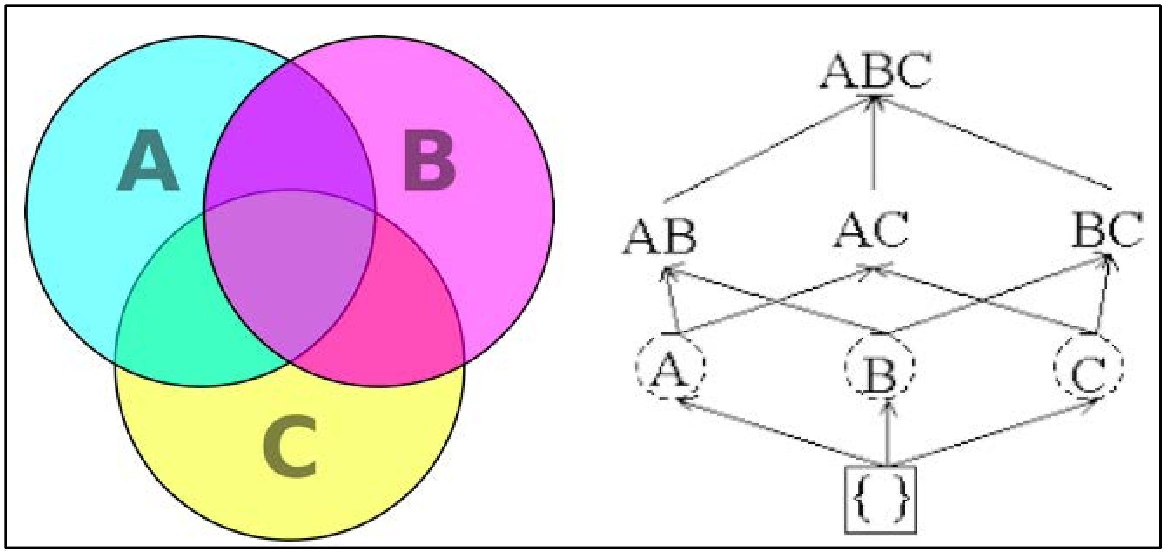


Figure 20: Overlapping Sub-problems and their Optimal Substructures

Current greedy algorithms solving the same problem can find models that give 70% accuracy. However, the Dynamic Programming Algorithm presented in this thesis provides a model C102 B120 A20 with 90% accuracy. This model is the best found in the whole search space and called optimal model. Optimal here means that the algorithm builds the UIUP tree that satisfies the maximum number of examples in the training dataset. It is optimal because any subtree S of the optimal tree is also optimal to the examples in E, which is the set of examples with every object restricted to the attributes in S. The optimal ordering of attributes A, B, and C is found using the calculations demonstrated in Table 9.

ABC	subset	A	B	C	F	Order
000	{}				0	
001	{C}			5	5	C102
010	{B}		3		3	B120
011	{B,C}		5 + F^{unseen} ({B}) = 5+2 = 7	3 + F^{unseen} ({C}) = 3+0 = 3	7	C102 B120
100	{A}	6			6	A102
101	{A,C}	5 + F^{unseen} ({A}) = 5+3 = 8		6 + F^{unseen} ({C}) = 6+1 = 7	8	C102 **A102**
110	{A,B}	3 + F^{unseen} ({A}) = 3+0 = 0	6 + F^{unseen} ({B}) = 6+1 = 7		7	A102 B12
111	{A,B,C}	7 + F^{unseen} ({A}) = 7 + 2 = 9	8 + F^{unseen} ({B}) = 8 + 0 = 8	7 + F^{unseen} ({C}) = 7 + 0 = 7	9	C102 B120 **A20**

Table 9: Dynamic Programming Memorization Table

The number of calls to algorithm 1 done by algorithm 2 equals the sum of all subsets' sizes in a power set, as presented in the below formula.

$$\sum_{i=0}^{n} \binom{n}{i} i = \sum_{i=0}^{n} \frac{n! \cdot i}{i!(n-i)!} = n \sum_{i=1}^{n} \binom{n-1}{i-1} = n2^{n-1}$$

The advantages of the dynamic programming algorithm are: its robustness against noise, hidden ties, and inconsistencies; it finds the most accurate UIUP models (optimal UIUP models). However, it has the following disadvantages: it's not feasible for learning UIUP trees of depth more than 26 (#attributes). It is bounded like Held-Carp algorithm for traveling salesman problem.

The pseudocode of the DPA algorithm is shown in Figure 21. It returns a model with the highest accuracy score. DPA requires the use of a helper function that generates a summary table from a given dataset. In Figure 21, *tsp* represents Algorithm 1 which is a modified function but similar implementation of Held-Karp algorithm for solving traveling salesman problem. Function $M(sT, a, t)$ is the function that prepares the graph matrix to *tsp* function and takes the summary table, one attribute, and seen examples to avoid then it returns a graph matrix. Figure 21 below presents the pseudocode of Algorithm 2.

```
    Input: $\mathcal{A}$ is the set of available attributes
    Output: Optimal Score , Optimal Model

Global Variables
    $sT$, Summary Table for Training Dataset
    $OptScore$, Optimal Scores Memory
    $OptOrder$, Optimal Orders Memory
end Global Variables

function DPA ( $\mathcal{A}$ )
    if $\mathcal{A}$ in $OptScore$ or $|\mathcal{A}| == 0$ then
        return $OptScore(\ \mathcal{A}\ )$ , $OptOrder(\ \mathcal{A}\ )$
    else
        for each $a \in \mathcal{A}$ do
            $t \leftarrow \mathcal{A} - (a)$
            $OptScore(\ t\ )\ ,\ OptOrder(\ t\ ) \leftarrow DPA(\ t\ )$
            $s\ ,\ p \leftarrow tsp(\ \mathcal{M}(\ sT\ ,a\ ,t\ )\ )$
            $children.append(\ (\ OptScore(\ t\ ) + s\ , OptOrder(\ t\ ) + `\ ' + a + p\ )\ )$
        end for
        return $max(children)$
    end if
end function
```

Figure 21: Pseudocode for Algorithm 2

4.2.1 Complexity Analysis

DPA is optimal in terms of the accuracy of the found model. Accuracy is normally the main concern in data mining and machine learning fields because it translates into correct predictions and good decisions. DPA finds the optimal model in the search space like any brute-force algorithm would do, but it is much faster than brute-force because it is enhanced by its inherent memorization technique that dramatically prunes the search space without losing states. DPA breaks the decision problem into smaller sub-problems with optimal solutions. Proof of its optimality follows from Bellman's principle of optimality; Any tree T` that is a subtree of an optimal tree T will make T` optimal to the same dataset per the principle of optimality.

Function tsp can be changed later to any faster algorithm that solves the Traveling Salesman problem. Hence, DPA algorithm complexity depends on tsp/algorithm 1 complexity. Let's assume *n* is the number of attributes, then DPA/Algorithm 2 will call tsp/algorithm 1:

$\frac{n}{2} * 2^n = n * 2^{\{n-1\}}$ times and the proof was provided in the previous session. Assume using Held-Karp Algorithm to implement tsp/algorithm 1 with $O(2^m m^2)$ for *m* nodes where *m* is the maximum number of values found in an attribute. Assume *E* is the set of examples in the training dataset. DPA's Time complexity = time to generate the summary table / Data Transformation + (complexity of Algorithm 1) * (#calls to Algorithm 1)

= Data Transformation time + $\left(2^m m^2 * n 2^{\{n-1\}}\right) = \left(n|E| + (2^m m^2)(n 2^{\{n-1\}})\right) =$

$$(n * |E|) + (2^{(n+m-1)} * n * m^2)$$

If |E| is extremely large, then it will dominate the time complexity leading the time complexity to be linear in |E|. |E| size can be extremely large, up to ${}^{\#Outcome}C_2$.

The space complexity for creating a global summary table once is $O(2^n * n * m^2)$ of memory then queries about graph matrixes can be obtained in 0(1). Here is an example for how this DPA algorithm can handle big data and what are its limits. If n=20 and m=10, then: a) #products = 10^20, b) 1000 Exabyte space will be required for products list, c) summary table space = 2.1GB captures any subset of 5 * 10^39 examples (10^20 Zettabytes), and d) the size of the UIUP model would be: 20 + 20*10 = 220 characters which can be used later to predict or classify any example of the 5 * 10^39 examples.

It is important to note that the DPA can be boosted by adding two important features. First, it could be implemented to run on parallel machines like Google Reduce Map. DPA is parallel-friendly because it is a dynamic programming algorithm so it can independently solve sub-problems and store their optimal-solutions. Second, DPA's algorithm 1 depends on the Traveling Salesman Problem (TSP) which currently has 30 nodes as upper bound. Any advancement or more efficient algorithms solving TSP would simply mean an advancement to DPA. If DPA uses any approximation-based algorithm for algorithm 1 that runs in polynomial time like voting, this will mean the DPA will increase its limit to handling thousands of attribute values instead of tens.

The real challenge facing future researchers solving the same problem will remain the same: what are the minimum number of states to explore before finding the optimal solution. The way to improve DPA could be via finding a better recursive formula for the dynamic programming algorithm to optimize. These improvements are left for future research.

4.3 Results

This section presents an empirical analysis of the two algorithms: the dynamic programming based algorithm (DPA), and the genetic algorithm (GA). To evaluate DPA and GA, the greedy algorithm is used as a baseline and an empirical analysis is performed on sets of examples given by hidden randomly generated LP-Tree. Synthetic data was used; the examples are produced from a specific hidden model and a percentage of examples were flipped to create inconsistent examples to emulate practical settings of noise, ties, and inconsistencies in real-world datasets. The greedy algorithm, used as a baseline, was developed by Liu [Liu15]. Time and accuracy were measured for different dataset sizes with different noise levels. Experiments were conducted on an processor "Apple MacBook Air Mid-2013 Model" computer with a 1.7 GHz Intel Core i7 and 8 GB of memory.

A domain of 10 attributes, each of 5 values, was used for the experiments. Thus, the universe contained 5^{10} objects, giving 5×10^{13} possible examples. At first, a random UIUP model of these attributes with random orderings as their local preferences, and a set D of random examples for training and testing, were generated. Then, set D was processed based on a noise percentage N where N · |D| examples are randomly selected and flipped. A dataset split of 80% for training and 20% for testing.

Experiment 1 tested the accuracy and time for domain of ten attributes and five values each, for N 15%, and for D of sizes 10^3, 10^4, ..., and 10^6. The instance for every D was repeated for five times and the averages were reported in the following figures.

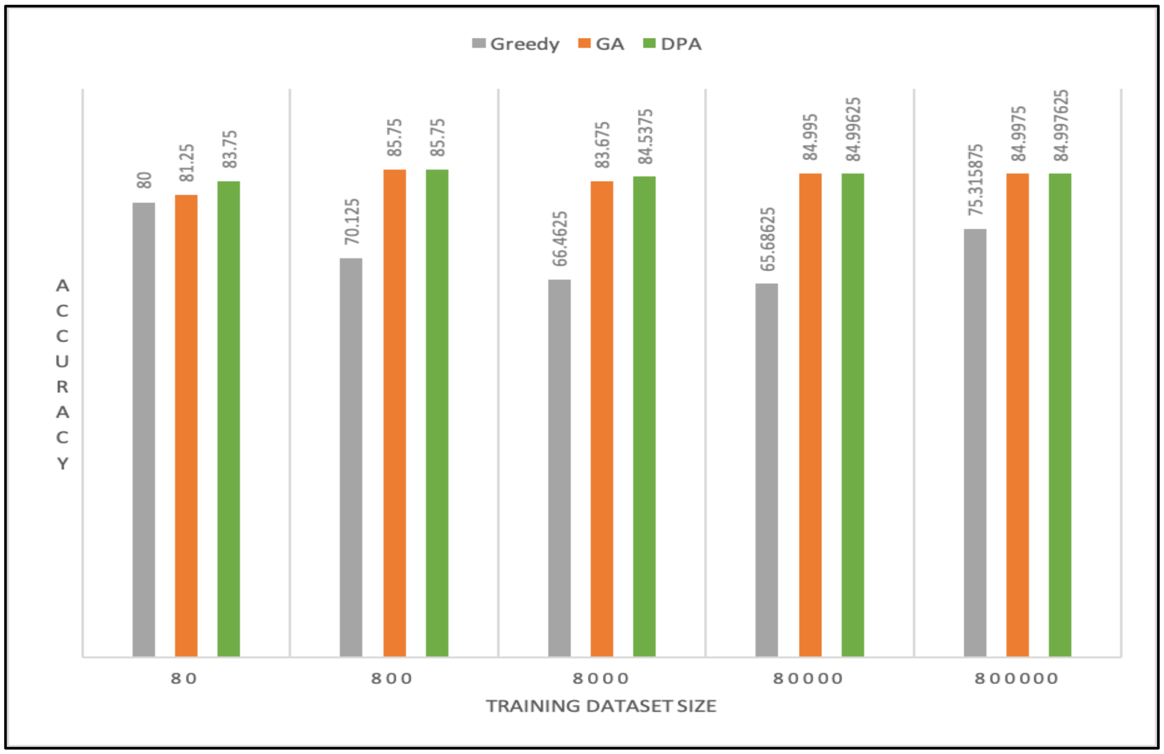


Figure 22: Accuracy Using Training Dataset

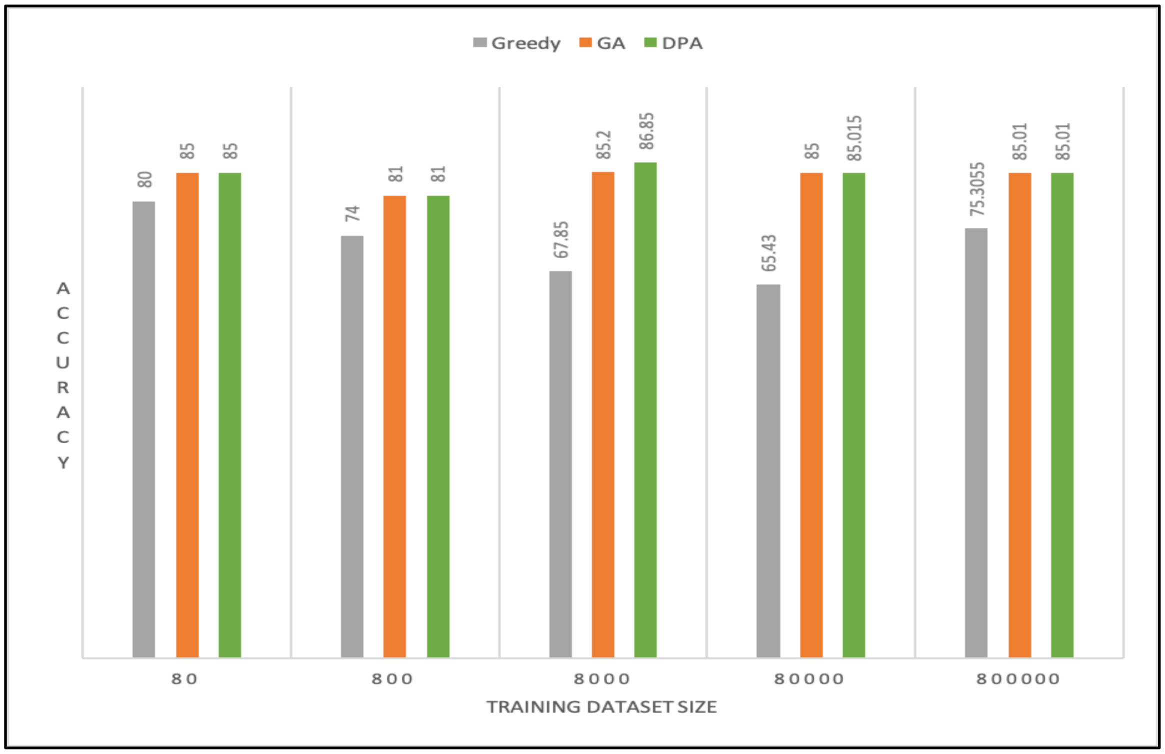


Figure 23: Accuracy Using Test Dataset

From Figure 22 and Figure 23, it is obvious that DPA obtained the highest accuracy on the training and testing examples. GA finished very close second, within 1% compared to DPA, while the greedy algorithm finished last. Figure 24 shows the total execution time, including both training and testing, for various training data sizes. Clearly, DPA, despite of 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 much larger constant in the asymptotic notion of Greedy than that of DPA. GA takes the most time as it goes through many generations.

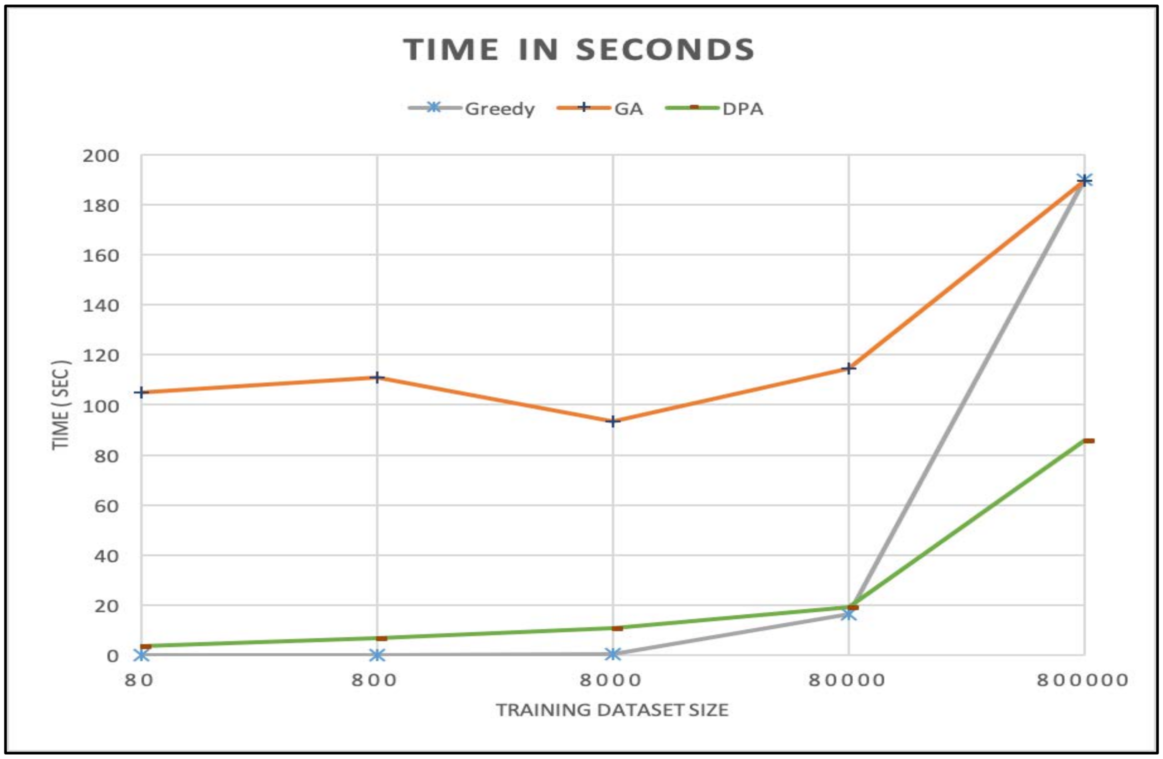


Figure 24: Time in Seconds to learn a model

Experiment 2 tested the effect of the training dataset size on accuracy. A domain of four attributes and four values each is used. Dataset sizes were percentages of the whole possible comparisons/strict examples such as 0%, 10%, 20%, and up to 100%. Two level of noises were tested for N of value 5% and 50%. The experiment was repeated 10 times and the averages were reported in Figure 25 and Figure 26. The generated noisy examples grew when the dataset size was increased. These figures clearly show that DPA was always able to reach to the most accurate model. For this reason, DPA is thought to give the optimal UIUP model followed by the GA, which gives a near-optimal UIUP model.

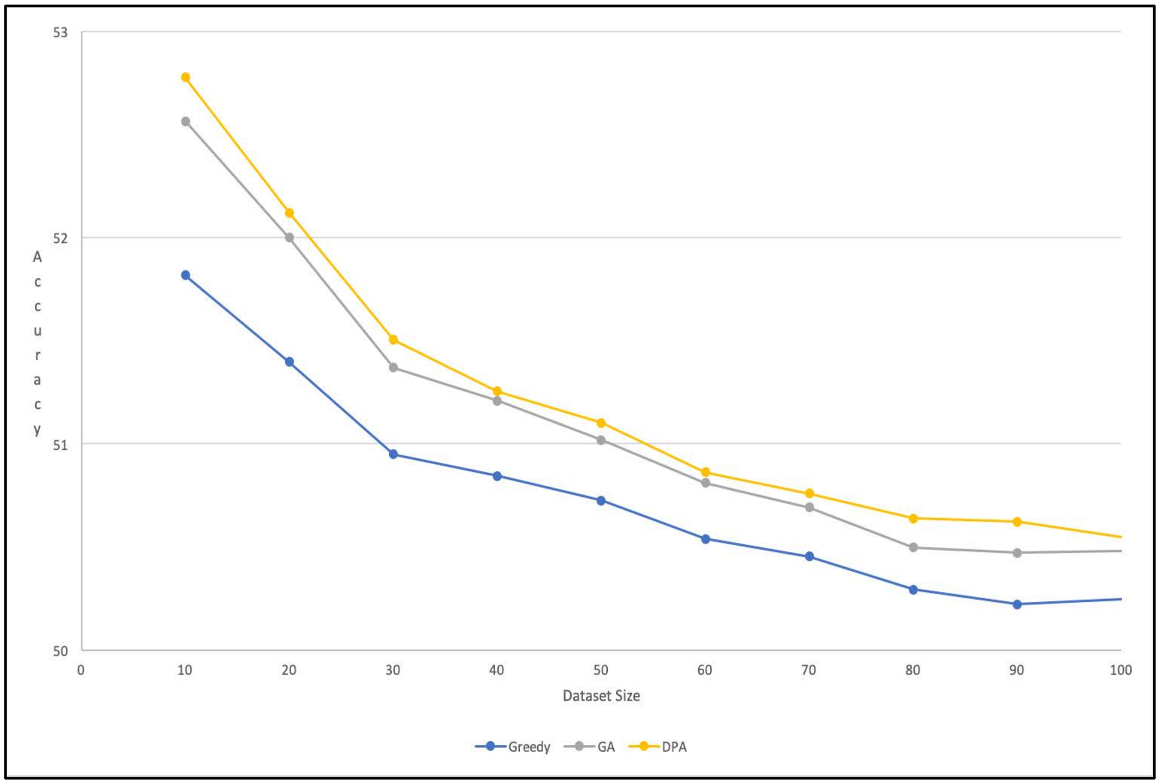


Figure 25: Noise in training dataset is 5 %

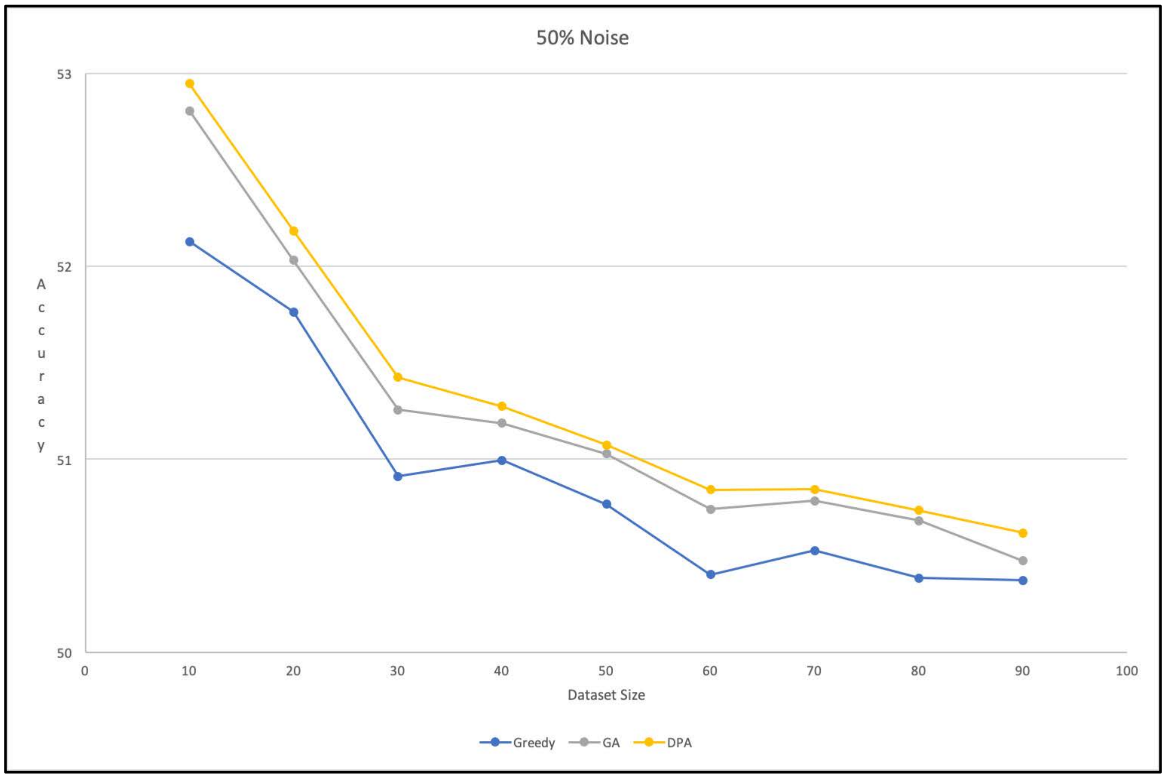


Figure 26: Noise in training dataset is 50 %

Other experiments were also performed. For example, one experiment was performed to test the effect of noise on the accuracy on a domain of four attributes and four values. It was intuitively concluded that accuracy decreases as the level of noise in the training dataset increases. However, the accuracy of DPA was always higher than both GA and Greedy. One last experiment was performed to test the effect of noise on the learning time and a domain of four attributes and four values was used. That experiment gave similar results to these of experiment two; GA was the slowest to learn a UIUP model which means DPA and Greedy would be fast regardless of the noise level in the dataset. One more advantage of DPA besides being fast, it always provides the most accurate UIUP model.

All experiments show that DPA is better than the genetic algorithm and better than the greedy one too in terms of accuracy and total execution time. DPA showed best accuracy results for preference learning while learning UIUP tree models from noiseless and noisy data. This algorithm is optimal in terms of accuracy but it has an upper bound limit. The computational complexity of DPA is exponential in terms of number of attributes and values. Ten attributes and ten values have been used and tested which can represent 10 billion different products. This can lead to 5e19 pairwise comparisons and any subset of these comparisons could be presented as a training dataset. Hence, the DPA can learn 1.4e72 different UIUP preference models given a domain of ten attributes and ten values. The DPA found the optimal models within less than one minute for ten attributes and ten values domains for a training dataset sizes of up to 10k.

Chapter 5

CONCLUSION

Making good decisions is sometimes critical, especially when bad decisions are costly. Understanding hidden preferences of one agent or a whole society can better guide humans or artificial decision makers. Better decision making is favored by managers or government officials and is also needed by systems such as recommendation agents or customizable e-commerce websites like Amazon. The preference problem is an important problem faced by decision makers. Therefore, preference understanding, visualization and learning are crucial research areas.

This research introduced a framework that helps agents visualize and understand preferences. Moreover, this research introduced two preference learning algorithms (GA and DPA) for the NP-hard problem named "MAXLEARN" by Liu [Liu15]. The space of this search problem is extremely huge and finding a good UIUP decision tree is challenging because the number of candidate solutions grows exponentially depending on the number of attributes, values, and products. The current best known algorithms are mixture of greedy and brute-force such as in [Boo10, Liu15]. Developing efficient, new algorithm for this type of problem is very useful and could lead to advances across disciplines.

For the contributions to the field of computer science, two new learning algorithms named GA and DPA were devised and introduced besides developing a framework that enable researchers and agents understand and visualize the field of preferences. The GA algorithm is a local-search learning algorithm that use genetic algorithms techniques while DPA uses dynamic programming. The framework would allow agents to experiment over identified domains such as cars to demonstrate an algorithm's effectiveness and the different preference trees. The framework gives agents the option to choose between XML, JSON, Outline, or Graphical representations.

To conclude, when users can only pick one product, and afford buying only one car for example, e-commerce faces uncertainty as of which products to recommend. Hence, new machine learning algorithms are needed for preference learning to advance decision making and smart targeted marketing. Decision makers from many industries such as retail, financial services, healthcare, e-commerce, and social media may benefit from these new algorithms. It is typical for decision makers to favor the more accurate algorithms because they reduce the number and hence cost of bad decisions.

REFERENCES

[Allen17]
Allen, Thomas E., Cory Siler, and Judy Goldsmith. "Learning tree-structured cp-nets with local search." Proceedings of the... International Florida Artificial Intelligence Research Society Conference. 2017.

[Booth10]
Booth, Richard, et al. "Learning conditionally lexicographic preference relations." ECAI. 2010.

[Boutilier04]
C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, D. Poole, CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. J. Artif. Intell. Res. 21, 135–191 (2004)

[Brandt16]
Brandt, Felix, et al., eds. Handbook of computational social choice. Cambridge University Press, 2016.

[Cohen99]
W.W. Cohen, R.E. Schapire, Y. Singer, Learning to order things. J. Artif. Intell. Res. 10, 243– 270 (1999)

[Coppersmith06]
D. Coppersmith, L. Fleischer, A. Rudra, Ordering by weighted number of wins gives a good ranking for weighted tournaments, in Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA-06) (2006), pp. 776–782

[Dunn18]
Dunn, Jack. Optimal Trees for Prediction and Prescription. Diss. PhD thesis, Massachusetts Institute of Technology, 2018. http://jack. dunn. nz/papers/Thesis. pdf, 2018.

[Dekel04]
O. Dekel, C.D. Manning, Y. Singer, Log-linear models for label ranking, in Advances in Neural Information Processing Systems (NIPS-03), ed. by S. Thrun, L.K. Saul, B. Schölkopf (MIT, Cambridge, MA, 2004), pp. 497–504

[Dewdney93]
A. K. Dewdney. “The New Turing Omnibus: Sixty-Six Excursions in Computer Science”, (July 1993)

[Feng15]
Feng, Tie, et al. "A compromise-negotiation framework based on Game theory for eliminating requirements inconsistency." Tehnički vjesnik 22.5 (2015): 1085-1092.

[Fürnkranz03]
J. Fürnkranz, E. Hüllermeier, Pairwise preference learning and ranking, in Proceedings of the 14th European Conference on Machine Learning (ECML-03), vol. 2837, Lecture Notes in Arti- ficial Intelligence, ed. by N. Lavracˇ, D. Gamberger, H. Blockeel, L. Todorovski (Springer, Cavtat, Croatia, 2003), pp. 145–156

[Fürnkranz10]
Fürnkranz, Johannes, and Eyke Hüllermeier. 2010. Preference Learning (1st ed.). Springer-Verlag, Berlin, Heidelberg.

[Fürnkranz11]
Fürnkranz, Johannes, and Eyke Hüllermeier. "Preference learning." Encyclopedia of Machine Learning. Springer, Boston, MA, 2011.

[Gansner93]
Gansner, Emden R., et al. "A technique for drawing directed graphs." IEEE Transactions on Software Engineering 19.3 (1993): 214-230.

[Gordon08]
S. Gordon, M. Truchon, Social choice, optimal inference and figure skating. Soc. Choice Welfare 30(2), 265–284 (2008)

[Haddawy03]
P. Haddawy, V. Ha, A. Restificar, B. Geisler, J. Miyamoto, Preference elicitation via theory refinement. J. Mach. Learn. Res. 4, 317–337 (2003)

[Har-Peled02]
S. Har-Peled, D. Roth, D. Zimak, Constraint classification: A new approach to multiclass classification, in Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT-02), ed. by N. Cesa-Bianchi, M. Numao, R. Reischuk (Springer, Lübeck, Germany, 2002), pp. 365–379

[Held62]
Held, Michael, and Richard M. Karp. "A dynamic programming approach to sequencing problems." Journal of the Society for Industrial and Applied Mathematics 10.1 (1962): 196-210.

[Joachims02]
T. Joachims, Optimizing search engines using clickthrough data, in Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02) (ACM, 2002), pp. 133–142

[Kaci11]
Kaci, Souhila. Working with preferences: Less is more. Springer Science & Business Media, 2011.

[Laukkanen02]
Laukkanen, Sanna, Annika Kangas, and Jyrki Kangas. "Applying voting theory in natural resource management: a case of multiple-criteria group decision support." Journal of Environmental Management 64.2 (2002): 127-137.

[Lewis-Beck11]
Lewis-Beck, Michael Steven, and Richard Nadeau. "Economic voting theory: Testing new dimensions." Electoral Studies 30.2 (2011): 288-294.

[Li12]
Li, Xinyu et al. “An active learning genetic algorithm for integrated process planning and scheduling.” *Expert Syst. Appl.* 39 (2012): 6683-6691.

[Liu15]
Liu, Xudong, and Miroslaw Truszczynski. "Learning Partial Lexicographic Preference Trees over Combinatorial Domains." AAAI. Vol. 15. 2015.

[Liu18]
Liu, Xudong, and Miroslaw Truszczynski. "Preference Learning and Optimization for Partial Lexicographic Preference Forests over Combinatorial Domains." International Symposium on Foundations of Information and Knowledge Systems. Springer, Cham, 2018.

[Medioni00]
Medioni, Gérard, Chi-Keung Tang, and Mi-Suen Lee. "Tensor voting: Theory and applications." Proceedings of RFIA. Vol. 2000. 2000.

[Mitchell99]
Mitchell, Melanie. An introduction to genetic algorithms. MIT press, 1998.

[Pigozzi16]
Pigozzi, Gabriella, Alexis Tsoukias, and Paolo Viappiani. "Preferences in artificial intelligence." Annals of Mathematics and Artificial Intelligence 77.3-4 (2016): 361-401.

[Procaccia13]
Procaccia, Ariel D. "How is voting theory really useful in multiagent systems." available online, URL: http://www. cs. cmu. edu/arielpro/papers/vote4mas. pdf (DOA: 15.01. 2013).

[Radlinski05]
F. Radlinski, T. Joachims, Learning to rank from implicit feedback, in Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD-05) (2005), pp. 239–248

[Russell03]
Russell, Stuart J, Peter Norvig, and John Canny. "Artificial Intelligence: A Modern Approach", 2003. Print.

[Sobolevsky14]
Sobolevsky, Stanislav, et al. "General optimization technique for high-quality community detection in complex networks." Physical Review E 90.1 (2014): 012811.

[Tesauro89]
G. Tesauro, Connectionist learning of expert preferences by comparison training. in Advances in Neural Information Processing Systems 1 (NIPS-88), ed. by D. Touretzky (Morgan Kaufmann, 1989), pp. 99–106

[Ukkonen09]
A. Ukkonen, K. Puolamäki, A. Gionis, H. Mannila, A randomized approximation algorithm for computing bucket orders. Inf. Process. Lett. 109 (2009)

[Wang94]
J. Wang, Artificial neural networks versus natural neural networks: A connectionist paradigm for preference assessment. Decision Support Syst. 11, 415–429 (1994)

[Xia11]
Xia, Lirong, "Computational Voting Theory: Game-Theoretic and Combinatorial Aspects". Duke University, 2011.

[Yaman10]
Yaman, Fusun, Thomas J. Walsh, and Michael L. Littman. "Learning lexicographic preference models." Preference learning. Springer, Berlin, Heidelberg, 2010.

VITA

Ahmed Moussa expects to receive a Master of Science in Computer and Information Sciences from the University of North Florida in April 2019. Dr. Xudong Liu of the University of North Florida is serving as Ahmed's thesis advisor. Ahmed is the recipient of a software patent by USPTO and the recipient of Upsilon Pi Epsilon's International Computing Honor Society Award. Ahmed participated in the annual ACM/ICPC USA southeast regional competitive programming contest, where over 80 colleges actively competed against each other in teams of three. He and his team ranked third in division II.

Ahmed has earned his B.S. in Computer Science from the American University in Cairo in 2011. He was part of Leadership for Education and Development Program during his undergraduate studies. In high school, Ahmed won a silver medal in the Egyptian Mathematics Olympiad. Ahmed will work for Amazon in Silicon Valley as Software Development Engineer upon graduation.