2021

# REPRESENTING AND LEARNING PREFERENCES OVER COMBINATORIAL DOMAINS

Michael Huelsman
*University of Kentucky*, michael.huelsman@gmail.com
Digital Object Identifier: https://doi.org/10.13023/etd.2021.139

# REPRESENTING AND LEARNING PREFERENCES OVER COMBINATORIAL DOMAINS

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for
the degree of Doctor of Philosophy
in the College of Engineering at the
University of Kentucky

By
Michael Andrew Huelsman
Lexington, Kentucky

Director: Dr. Miroslaw Truszczynski, Professor of Computer Science
Lexington, Kentucky 2021

ABSTRACT OF DISSERTATION

# REPRESENTING AND LEARNING PREFERENCES OVER COMBINATORIAL DOMAINS

Agents make decisions based on their preferences. Thus, to predict their decisions one has to learn the agent's preferences. A key step in the learning process is selecting a model to represent those preferences. We studied this problem by borrowing techniques from the algorithm selection problem to analyze preference example sets and select the most appropriate preference representation for learning. We approached this problem in multiple steps.

First, we determined which representations to consider. For this problem we developed the notion of preference representation language subsumption, which compares representations based on their expressive power. Subsumption creates a hierarchy of preference representations based solely on which preference orders they can express. By applying this analysis to preference representation languages over combinatorial domains we found that some languages are better for learning preference orders than others.

Subsumption, however, does not tell the whole story. In the case of languages which approximate each other (another piece of useful information for learning) the subsumption relation cannot tell us which languages might serve as good approximations of others. How well one language approximates another often requires customized techniques. We developed such techniques for two important preference representation languages, conditional lexicographic preference models (CLPMs) and conditional preference networks (CP-nets).

Second, we developed learning algorithms for highly expressive preference representations. To this end, we investigated using simulated annealing techniques to learn both ranking preference formulas (RPFs) and preference theories (PTs) preference programs.

We demonstrated that simulated annealing is an effective approach to learn preferences under many different conditions. This suggested that more general learning strategies might lead to equally good or even better results. We studied this possibility by considering artificial neural networks (ANNs). Our research showed that ANNs can outperform classical models at deciding dominance, but have several significant drawbacks as preference reasoning models.

Third, we developed a method for determining which representations match which example sets. For this classification task we considered two methods. In the first method we selected a series of features and used those features as input to a linear feed-forward ANN. The second method converts the example set into a graph and uses a graph convolutional neural network (GCNN). Between these two methods we found that the feature set approach works better.

By completing these steps we have built the foundations of a portfolio based approach for learning preferences. We assembled a simple version of such a system as a proof of concept and tested its usefulness.

KEYWORDS: Preference Representation, Preference Reasoning, Preference Learning

Author's signature: Michael Andrew Huelsman

Date: May 13, 2021

# REPRESENTING AND LEARNING PREFERENCES OVER COMBINATORIAL DOMAINS

By
Michael Andrew Huelsman

Director of Dissertation: Miroslaw Truszczynski

Director of Graduate Studies: Zongming Fei

Date: May 13, 2021

To Leslie W. Simpson, Janice C. Simpson, Beverly L. Simpson, Robert W. Huelsman, Tyler Arnold, William Bailey, Izzy Boyer, Antonnia Jackson, Ellen Johnson, Jesse Johnson, Mason Lilly, Frances Meredith, Jacob Schlueter, Cory Siler, and Rheannon Townsend:

Thank you.

## ACKNOWLEDGMENTS

The following dissertation, while an individual work, benefited from the insights and direction of several people. First, my Dissertation Chair, Dr. Miroslaw Truszczynski, exemplifies the high quality scholarship to which I aspire. In addition, Dr. Truszczynski provided timely and instructive comments and evaluation at every stage of the dissertation process, allowing me to complete this project on schedule. Next, I wish to thank the complete Dissertation Committee, and outside reader, respectively: Dr. Judy Goldmsith, Dr. Brent Harrison, Dr. Samson Cheung, and Dr. Bert Guillou. Each individual provided insights that guided and challenged my thinking, substantially improving the finished product. In addition to the technical and instrumental assistance above, I received equally important assistance from family and friends. My mother, Beverly Simpson, whose consistent encouragement was met with begrudging gratitude. Mason Lilly, who spent many late nights listening to me rambling on and on about my research and provided a good sounding board. Finally, all members of the BCDS who managed to keep me sane throughout this process.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1 Introduction

Preferences are an important topic of study in several fields such as economics, philosophy, psychology, marketing, and computer science. Each studies preferences in a somewhat different way and with different methods. In computer science, the questions of interest revolve around preference representation and preference reasoning algorithms. Applications of preference reasoning and representation range from database design to decision support tools to recommender systems [Domshlak et al., 2011]. This work focuses on the interrelation of different methods of representing preferences and how this affects the preference learning problem.

As an example of a potential preference related problem, suppose we are trying to find the perfect gift for a friend. Since our friend happens to need a new computer we would like to get them one. While we could simply buy the computer we would like, it is likely that we have different preferences from our friend. The first problem to tackle in this scenario is one of representation. How are we going to represent our friend's preferences? A great deal of work has gone into defining and studying various languages for representing preferences [Kaci, 2011]. Once we have selected a preference representation language we must build a model of that language to represent our friend's decision making process. How to build such a model is our second problem. We could ask our friend their opinion directly, but too many pointed questions might give away that we are trying to buy them a computer, and gift giving implies a level of secrecy. This means it is preferable to not directly ask questions, but build our model based on decisions we know our friend has made in the past between different computers.

This example introduces two key preference reasoning problems. The first is language selection, i.e. how are we representing preferences. The second is preference learning, i.e. how do we build accurate preference models. In this work we approach the langauge selection problem by first creating a theoretical framework for understanding how expressive a language is, in relation to other languages. From there we develop systems which take as input a collection of pairwise comparisons and classify them as being created by one of several languages. As for the problem of preference learning, we develop a method of learning preferences using simulated annealing. Using these learning methods along with our work on language selection we create a preference learning system which first analyzes the provided preference examples and then selects the best algorithm to use for that specific collection of examples. This approach works better than simply always selecting the same language, even if that language is very expressive.

## 1.1 Our Contributions

This work contributes to the understanding of the relations between preference representation languages, with a specific focus on those for expressing preferences over combinatorial domains, and of how these relations affect the task of preference learning. We define the concept of preference language subsumption which allows us to order preference representation languages based on their expressive power. We define expressive power to mean the class of preorders which a particular preference representation language is capable of producing.

Since subsumption deals with expressiveness, we expect that learning preferences using a more expressive language will produce better, or at least similar, results to using a less expressive language. We tested the impact of subsumption in learning using two languages, ranked preference formulas (RPFs) and preference theories (PTs). Learning a particular model of either the RPF or PT languages which achieves a specific threshold of accuracy is NP-complete, which we prove later, for both single and multi-agent settings. To overcome the inefficency of solving an NP-hard problem we learn approximations of the optimal models for RPFs and PTs. These models are learned using the local search technique of simulated annealing. We show the efficacy of this learning method by comparing it to a learning algorithm from the literature for learning lexicographic preference models (LPMs) [Schmitt and Martignon, 1999] which has good accuracy guarantees. Throughout our experiments we found that PTs have good accuracy on the widest variety of example sets. We expect such results given that PT is the most expressive language we consider.

During our learning experiments we also found that when learning from an example set of a model of a (unknown) language there is an increase in learning accuracy when learning a model of that language, i.e. if a set of examples was produced by an LPM then learning an LPM will give us the best learning performance. This trend implies that if one could identify, from a set of examples, the language of the model that produced it then learning a model of that language using the example set would increase the accuracy of our learned model.

While useful the subsumption relations has some limitations which must be stated. It may be the case that two languages are incomparable with respect to subsumption (neither is subsumed by the other), yet it may still be possible to use models from one language as useful approximations of models from another. In particular, as we will see in more detail later, we were able to find two languages, conditional preference networks (CP-nets) and conditional lexicographic preference models (CLPMs), where one language provides a strong approximation of the other. In this case, a CLPM can produce an over approximation of a CP-net's preference relation. This strong ability to approximate CP-net models using CLPMs is not indicated by subsumption. This is one of subsumption's primary analytical limitations.

In the settings used in this work the problem of finding the best model of a language (of the languages we learn) for a given example set is NP-hard. Given our success at using the metaheuristic of simulated annealing we considered the usefulness of another powerful metaheuristic, namely artifical neural networks (ANNs), for learning preferences. We tested using ANNs to learn qualitative preference and found that in the single-agent setting accuracy was high. While this approach showed promise there are some problems with the ANN approach. First, it is difficult to extract more nuanced information from ANNs, for example ANNs give no indication as to which features of an alternative might be important. Secondly, it is possible for an ANN, as we construct them, to learn contradictory information such as $A \succ B \succ C \succ A$.

The problem of preference learning, for all the languages we consider, does not increase in complexity when moving from a single-agent setting to a multi-agent setting. We applied our learning methods to multi-agent settings using two different criteria: utilitarian and maximin. The utilitarian criterion counts the number of satisfied examples, without considering which agent those examples came from. In simple terms, learning with the utilitarian criterion tries to maximize the number of total examples satisfied. On the other hand, the maximin criterion is the proportion of examples satisfied for the agent with the smallest proportion of their example satisfied. We found that when learning using the utilitarian criterion our learned models performed similarly to how they performed in the single-agent setting. Learning using the maximin criterion favored the learning of a model from a more general langauge.

Our preference learning experiments indicate it is beneficial to know the language which includes the agent's preference model. In other words, if we can classify an example set by the language of the model used to generate it then we can improve our learning accuracy. We test two methods for solving this classification problem. First, we chose features to represent an example set, in our case the feature vector consisted of the proportion of the various possible pairwise relations in the example set. We then trained a neural network classifier, which was able to classify example sets with accuracy greater than 50%, on average, and much better than chance. Unfortunately, expanding this feature vector with additional information did not yield better results.

The other approach we tried was to use *graph convolutional neural networks* (GCNNs) as the basis for our classifier. Example sets were converted into relation graphs and then used as input for our GCNN models. While there has been previous success using GCNNs for graph classification it appears that such models do not perform well for our purposes. In most settings we achieved performance which was not significantly higher than guessing. Our results indicate that using GCNNs was worse than our feature vector approach, in terms of predictive accuracy. Using this information we constructed and tested a small porfolio-based preference learning system.

## 1.2 Related Works

This work deals with several preference representation languages. The languages we consider, from the literature, are lexicographic preference models (LPMs) [Fishburn, 1974], lexicographic preference trees (LP-trees) [Booth et al., 2010], conditional preference networks [Boutilier et al., 2004], conditional lexicographic preference models (CLPMs) [Huelsman and Truszczyński, 2020], utility functions [Von Neumann and Morgenstern, 1944], weighted average utility functions [Debreu, 1954], penalty logic [Haddawy and Hanks, 1992], and answer set optimizations (ASOs) [Brewka et al., 2003]. This work expands those works by showing the interrelation of those language using the subsumption relation. While some of this work has already been done [Booth et al., 2009, Kohli and Jedidi, 2007, Wilson, 2009], the results are limited in scope and concern relations between similar languages. Our work with the subsumption relation provides general definitions and proof techniques and expands previous analysis to a greater number of languages. It is noteworthy that our case study does not only consider the presence of subsumption relations, which is how it has been discussed previously in the literature, but also the lack of subsumption relations. That is, we present and prove negative results showing no subsumption relations between pairs of languages.

Our other case study, involving the approximation of CP-net dominance using CLPMs, borrows the central approximation technique from Ahmed and Mouhoub [2018]. Our particular version differs in that we consider multiple models and remove the assumption that incomparability must be removed from a preference ordering. Additionally, we borrow a method of evaluating grouped lexicographic preferences from Yaman et al. [2008], although where their models derive equality ours derives incomparability. Our primary contribution consists of a method for using several CLPMs to approximate a single CP-net. This approximation works identically to the possibilistic approach proposed by Dubois et al. [2013]. However, they do not study any algorithmic methods to compute this approximation, which is central to our work. We empirically analyze the effectiveness of the approximation.

Learning preferences using search techniques is not new. A recent example of this is using local search to learn tree-structured CP-nets [Allen et al., 2017]. We are unaware of these techniques being applied to logical preferences and specifically ranked preference formulas (RPFs) and preference theories (PTs). When it comes to the use of neural networks for qualitative preference learning we are not aware of any related studies, but there has been previous work done which uses common machine learning techniques for preference learning purposes. Two major examples of this are the use of K-nearest neighbor in recommender systems [Habin and Vlado, 2007] and support vector machines in $SVM^{\text{rank}}$ [Joachims, 2002]. The key difference between those systems and our system is that they rely (implicitly or explicitly) on quantitative preferences while our approaches deal with qualitative preferences. Our primary point of comparison for the effectiveness of our learning methods is given by Schmitt and Martignon [2006], which details a greedy method for learning LPMs from a set of examples.

Our learning results indicate that one can learn a better model from an example set if one knows what language was used to generate it. This indicates that a system which selects a learning algorithm on a case-by-case basis would have improved learning accuracy. The general technique of analyzing a problem and then selecting the best algorithm to solve it is not new. This is generally referred to as the algorithm selection problem [Rice, 1976] and has been studied in various contexts. Our approach was inspired by the portfolio method of solving the algorithm selection problem [Leyton-Brown et al., 2003] and its successful application to both SAT solving [Xu et al., 2008] and answer set programming [Gebser et al., 2011].

# Chapter 2 Background

In this section we formally define the concepts underpinning this work. Specifically, we define the terms such as preference relations, preference ordering, combinatorial domain, preference representation language, and various preference reasoning problems. Alongside these definitions we provide examples and elaboration as needed.

## 2.1 Relations and Orders

Consider someone looking to buy a laptop. Suppose the decision has been narrowed to three possible choices: $a, b,$ and $c$. The buyer's preferences are expressed by how much they are willing to spend on each option such that the more the buyer is willing to spend the more preferred a particular laptop. This induces a preference order over the laptops. Of course, our example implies other inferences such as the degree to which one laptop is preferred to another. For example, if the buyer is willing to pay twice as much for $a$ over $b$, but the spending difference between $b$ and $c$ is minimal, we can infer that the buyer might be indifferent about receiving $c$ rather than $b$, but receiving $b$ instead of $a$ is a significant disappointment. Such assessments are easy when dealing with real number values for objects, but often we do not have such concrete information. Perhaps we simply know that $a$ is preferred to $b$ and $b$ is preferred to $c$, without any indication of the degree of preference between the alternatives. The former example represents quantitative preferences while the later represents qualitative preferences. This work considers problems dealing with qualitative preferences.

Consider an agent that can compare alternatives, that is, given two alternatives $\alpha$ and $\beta$, the agent can say whether $\alpha$ is at least as preferred as $\beta$. This defines the agent's *weak preference relation*, $\succeq$. In most studies of qualitative preference representation and reasoning it is assumed that weak preference relations form preorders. Formally, a binary relation $\succeq$ on a set of objects (outcomes, alternatives) $\Omega$ is reflexive if for any object $\alpha \in \Omega$, $\alpha \succeq \alpha$. It is transitive if for any three objects $\alpha, \beta, \gamma \in \Omega$ if $\alpha \succeq \beta$ and $\beta \succeq \gamma$ then $\alpha \succeq \gamma$. A binary relation $\succeq$ on $\Omega$ is a *preorder* if it is reflexive and transitive. As noted, weak preference relations are preorders.

In some settings, an agent compares alternatives in a strict manner, that is, given two alternatives $\alpha$ and $\beta$, the agent states whether $\alpha$ is strictly preferred to $\beta$. This defines the agent's *strict preference relation*, $\succ$. Strict preference relations are typically modeled using strict orders. A strict order, often referred to simply as an order, is defined as any binary relation which is *irreflexive* and *transitive*. Formally, a binary relation $\succ$ on a set of objects $\Omega$ is irreflexive if for any object $\alpha \in \Omega$, $\alpha \not\succ \alpha$, i.e. no alternative is strictly preferred to itself. One can show that for every strict order and two different alternatives $\alpha$ and $\beta$, if $\alpha \succ \beta$ then $\beta \not\succ \alpha$.

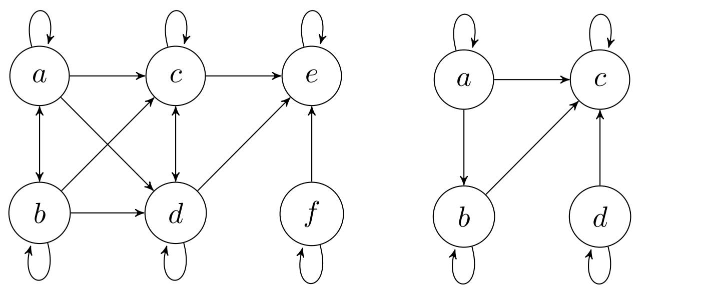
Given an agent's weak preference relation $\succeq$ one can derive several related relations. We define *preferential equivalence* or *indifference*, denoted $\approx$, by setting $\alpha \approx \beta$ if $\alpha \succeq \beta$ and $\beta \succeq \alpha$, which means $\alpha$ and $\beta$ are equally preferred. Preferential equivalence, by its definition, is symmetric and thus $\approx$ represents an equivalence relation because it is reflexive, symmetric, and transitive. We can also define *incomparability*, denoted $\bowtie$, by setting $\alpha \bowtie \beta$ if $\alpha \not\succeq \beta$ and $\beta \not\succeq \alpha$, which means $\alpha$ cannot be compared to $\beta$. Given a preorder $\succeq$ and another preorder $\succeq'$ if for all pairs $(\alpha, \beta) \in \Omega$, $\alpha \succeq \beta$ implies $\alpha \succeq' \beta$ then we say that $\succeq'$ *extends* $\succeq$.

A strict order can be dervied from a given preorder by first grouping alternatives based on their equivalence classes wrt the $\approx$ relation derived from $\succeq$. For two different equivalence classes $\boldsymbol{\alpha}, \boldsymbol{\beta} \subseteq \Omega$ we define the strict order $\succ$ by setting $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$ if there exists $\alpha \in \boldsymbol{\alpha}$ and $\beta \in \boldsymbol{\beta}$ such that $\alpha \succeq \beta$. For example if $\Omega = \{a, b, c\}$ and we have $a \succeq c$, $b \succeq c$, $a \succeq b$, and $b \succeq a$ (ignoring assumed reflexive comparisons) then the derived strict order is $\boldsymbol{\alpha} \succ \boldsymbol{\beta}$, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the equivalence classes defined by

$\succeq$ with $\boldsymbol{\alpha} = \{a, b\}$ and $\boldsymbol{\beta} = \{c\}$.

For a given (pre)order if for any two different alternatives $\alpha$ and $\beta$ either $\alpha \succeq \beta$ or $\beta \succeq \alpha$ then that (pre)order is called a *total* (pre)order. A total strict order is often called a *linear* order.

Figure 2.1: An example of the graphical representation of orders and preorders. Some edges implied by transitivity are not shown for simplicity.



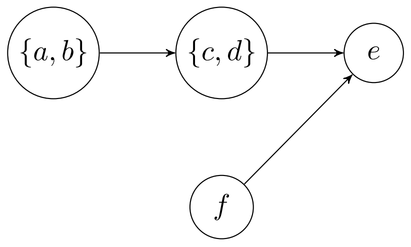
(a) Graph representation of a preorder.

(b) Graph representation of an order.

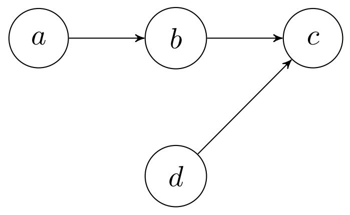
Any binary relation can be expressed as a graph. Given a binary relation $R$ on a set of elements $\Omega$, its corresponding graph $G = (\Omega, E)$ is defined by setting $(a, b) \in E$ if and only if $aRb$. This allows us to illustrate preorders in a graphical manner, see Figure 2.1a. Since we know that preorders are transitive and reflexive, one does not need to represent explicitly all the transitive or reflexive edges. A smallest set of edges that suffices to reconstruct a preorder is well defined. It gives rise to what is called the *Hasse* diagram. Consider the relation graphs given in Figures 2.1a and 2.1b. These relation graphs can be reduced to their Hasse diagrams given in Figures 2.2a and 2.2b, respectively.

Total preorders can be represented as a ranking. A ranking is a function $r$ which maps a set of objects $\Omega$ onto the natural numbers such that for every alternative $\alpha \in \Omega$, $r(\alpha) \geq 1$, and there exists an alternative $\beta \in \Omega$ such that $r(\beta) = r(\alpha) - 1$, except when $r(\alpha) = 1$. If the set of alternatives $\Omega$ is finite then any possible total preorder over it can be represented by some ranking function $r$. For a given ranking

Figure 2.2: Simplified versions of the graphs shown in Figure 2.1. Edges implied by transitivity and reflexivity not shown for simplicity. Equivalent alternatives have been collapsed into a single vertex.



(a) Graph representation of a preorder.



(b) Graph representation of an order.

function $r$ we define its corresponding preference relation $\succeq_r$ by setting $\alpha \succeq_r \beta$ if $r(\alpha) \leq r(\beta)$, thus lower ranked alternatives are preferred to higher ranked ones. For example if $\Omega = \{a, b, c, d\}$ then an example of a ranking function is:

$$r(\alpha) = \begin{cases} 1 & \alpha = a \\ 2 & \alpha = b \\ 3 & \alpha = c \text{ or } \alpha = d \end{cases}$$

This ranking function models the preorder $a \succeq b \succeq \{c, d\}$, where $\{c, d\}$ means that $c$ and $d$ are equally preferred. Often it is important to talk about the set of all objects with a particular rank. To this end, we define $\Omega_i = \{\alpha | \alpha \in \Omega, r(\alpha) = i\}$. A ranking represents a linear order when $|\Omega_i| = 1$ for each rank $i$.

## 2.2 Combinatorial Domains

A combinatorial domain is determined by a set of *attributes*, $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, and their domains $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_n\}$. Each attribute domain is finite and contains more than one value. We use the notation $\mathcal{D}(v_i)$ to denote the domain of an attribute $v_i$. The combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ defined by $\mathcal{V}$ and $\mathcal{D}$ is the cartesian product of individual attribute domains: $\mathcal{C}(\mathcal{V}, \mathcal{D}) = \mathcal{D}(v_1) \times \mathcal{D}(v_2) \times \ldots \times \mathcal{D}(v_n)$. Informally, a combinatorial domain is a collection of objects that can be abstracted to a tuple of values of attributes, one value per attribute. To avoid ambiguity in the case when domains of different attributes overlap, we commonly implicitly assume a fixed enumeration order of attributes. It allows us to recover the correct value of all attributes of an element from the tuple used to describe it. In some settings, when dealing with an alternative $\alpha \in \mathcal{C}(\mathcal{V}, \mathcal{D})$, we may find it useful to project $\alpha$ onto a subset of its attributes. We denote such a projection as $\alpha[A]$ for some set of attributes $A \subseteq \mathcal{V}$. In the case where we are projecting $\alpha$ onto a single attribute we use the notation $\alpha[a]$ rather than $\alpha[\{a\}]$ for simplicity.

Combinatorial domains are common in the real world. For example, the set of computers currently available for purchase is a combinatorial domain. In this case, attributes are properties such as the amount of RAM, the amount of space on the hard drive, the type of graphics card, and the type of CPU. Each of these attributes takes on one of several options as a value, and we can accurately describe a singular computer by specifying a value for each attribute. Similar combinatorial domains can be specified for apartments, cars, meals, etc. While it may not be readily apparent, combinatorial domains are common in one's day-to-day life.

The size of a combinatorial domain is determined by the number of attributes and size of the attribute domains. In general, the size of a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ is $\prod_{v \in \mathcal{V}} |\mathcal{D}(v)|$. In the real world attribute domains tend to be of different sizes. In theoretical studies it is common to assume that all domains are binary, this defines the subset of combinatorial domains called *binary combinatorial domains*. This is done because binary combinatorial domains are often easier to deal with and it is possible to transform any non-binary combinatorial domain into a binary combinatorial domain. Binary combinatorial domains are of size $2^n$, where $n$ is the number of attributes. This means that combinatorial domains are exponential in size, wrt the number of attributes.

Any finite set $O$ gives rise to a corresponding binary combinatorial domain $\mathcal{C}(O)$.

Each element in $o \in O$ serves as an attribute with the binary domain $\{o \text{ out}, o \text{ in}\}$, or $\{0, 1\}$. For example if $O = \{a, b, c, d\}$ then we can describe the alternative $\{a, c\}$ as the tuple $(1, 0, 1, 0)$ (assuming that the order of elements as attributes is $(a, b, c, d)$). This combinatorial domain is the same as the power set of $O$.

## 2.3 Preference Representation

A straightforward method of preference representation is to list the comparisons between each pair of objects in a particular domain. This type of representation is fine for small domains with a few objects, but in the worst case we may need $\mathcal{O}(n^2)$ relations, where $n$ is the number of alternatives, if we cannot assume relations by transitivity. In the case of linear orders we only need $\Theta(n)$ relations since we can simply enumerate alternatives from most preferred to least preferred. Of course even in the linear order setting if the domain is large then such a listing will not be space efficient. Since we deal with combinatorial domains, which are exponential in size wrt the number of attributes, listing relations is not efficient in either case. To overcome this difficulty we need methods of encoding preorders which are compact in size wrt the domain of alternatives. A side effect of having a compact representation is that we limit the class of preorders that can be represented. A *preference representation language* consists of a syntax that defines what statements are valid in the language and a semantics that specifies how to interpret those statements to induce a preference order. There are many preference representation languages, later referred to as preference languages or just simply languages, some of which we will introduce here.

Formally, a preference representation language is a collection of formal expressions (formulas, sequences of formulas, trees, etc.) We call these formal expressions *models*. The semantics of the language is a mapping from expressions of the language to preference orders over some combinatorial domain. This definition of a preference representation language is intentionally vague to allow for the variety of ways preferences can be represented. Each model from a language induces a relation; those models in the language that induce preorders can be used as representations of preference relations. For some languages all models induce preorders.

We now define several useful relations between models and between languages. One important relation is *model equivalence*. We consider two models to be equivalent, denoted $\equiv$, when the relations induced by the models are identical. In particular, if two models are equivalent, they express preference over the same domain of alternatives. However, model equivalence does not require that the models belong to the same language. Model equivalence allows us to define the key relation in our study, *language subsumption*. A language $\mathcal{L}$ subsumes another language $\mathcal{L}'$ if for each model $M' \in \mathcal{L}'$ there exists a model $M \in \mathcal{L}$ such that $M \equiv M'$. Language subsumption allows us to build a hierarchy of languages based on the preference orders they can express.

Languages fall into one of two broad categories: qualitative and quantitative. As we noted earlier this work limits itself to qualitative languages. A simple example of a qualitative preference representation is given by *preference formulas* [Kaci, 2011]. Consider a binary combinatorial domain $\mathcal{C}(\mathcal{V})$ and a language $\mathcal{L}(\mathcal{V})$ of propositional logic defined over the set $\mathcal{V}$ of attributes of the domain. There is a direct one-to-one correspondence between alternatives in $\mathcal{C}(\mathcal{V})$ and truth assignments to atomic formulas in the language $L(\mathcal{V})$. Namely, if an alternative $\alpha$ has value 1 for a particular attribute $v_i \in \mathcal{V}$ then the propositional variable $v_i$ is true in the truth assignment associated with $\alpha$. A formula $\varphi$ in the language $L(\mathcal{V})$ can be thought of as expressing a property of truth assignments and so, as a property of alternatives. Namely, a truth assignment (alternative) $\alpha$ has property $\varphi$ if $\alpha$ satisfies $\varphi$, or in other words is $\varphi$ true under the truth assignment induced by $\alpha$; we write $\alpha \models \varphi$ to denote this. If $\varphi$ denotes a desirable property for alternatives, we can take $\varphi$ as describing a preference order among alternatives. Namely, we can regard all alternatives having property $\varphi$

as equivalent, all alternatives not having property $\varphi$ as equivalent too, and the former as strictly preferred to the latter. Formally we define the preference relation $\succeq_\varphi$ as follows: $\alpha \succeq_\varphi \beta$ if $\alpha \models \phi$ and $\beta \models \varphi$; or $\alpha \not\models \varphi$ and $\beta \not\models \varphi$; or $\alpha \models \varphi$ and $\beta \not\models \varphi$.

Consider the combinatorial domain defined by treating elements of the set $O = \{a, b, c, d\}$ as propositional variables, formulas over $O$ express properties of alternatives (subsets of $O$). An example of a formula is:

$$\varphi = (\bar{a} \wedge b) \vee c.$$

In the corresponding preference order $\succeq_\varphi$ alternatives that satisfy the formula are preferred to alternatives that do not. In this example the subset $\{b\}$ (the alternative $(0, 1, 0, 0)$) is preferred to the subset $\{a, d\}$ (the alternative $(1, 0, 0, 1)$). In the formalism of preference formulas, models are given by a single formula and so the preference orders that can be expressed are limited to total orders composed of at most two clusters of equally preferred alternatives. In this example, the top (preferred) cluster consists of all alternatives that have a 1 in the third position (contain $c$) or that have 0 in the first position and 1 in the second position (contain $b$ and do not contain $a$), with all other alternatives in the bottom cluster.

Instead of a single single formula, we can consider sets of formulas. In this case we can assign each formula a weight, thus prioritizing properties based on their importance. An example of such a language is *penalty logic* [Haddawy and Hanks, 1992].

Penalty logic is a quantitative preference representation where preferences are represented by a set of propositional formulas $\boldsymbol{\varphi} = \{\varphi_1, \varphi_2, \ldots, \varphi_m\}$ and a set of positive weights $\boldsymbol{W} = \{w_1, w_2, \ldots, w_m\}$. The weights measure the desirability of having the corresponding formula satisfied by an alternative. Specifically, we define the utility of an alternative by setting:

$$u(\alpha) = \sum_{i:\alpha \not\models \varphi_i} -w_i.$$

The preference relation $\succeq_{\boldsymbol{\varphi},\boldsymbol{W}}$ induced by a given model of the penalty logic language is defined by setting $\alpha \succeq_{\boldsymbol{\varphi},\boldsymbol{W}} \beta$ if $u(\alpha) \geq u(\beta)$. Since weights may differ from formula to formula, an alternative that fails to satisfy many low weight formulas may be preferred to an alternative that fails to satisfy only one formula, assuming that formula has a sufficiently large weight. An example of this type of preference representation, using the combinatorial domain over the set $O = \{a, b, c, d\}$ we used earlier, is

$$\boldsymbol{\varphi}, \boldsymbol{W} = \begin{cases} \varphi_1 = (\bar{a} \wedge b) \vee c & w_1 = 10 \\ \varphi_2 = \bar{a} & w_2 = 3 \\ \varphi_3 = \bar{c} \wedge d & w_3 = 20 \end{cases} .$$

This example assigns the value $-33$ to the subset $\{a, b\}$ and the value $-20$ to the subset $\{c\}$ implying that $\{c\} \succ_{\boldsymbol{\varphi},\boldsymbol{W}} \{a, b\}$. Penalty logic uses weighted propositional formulas to define a *utility function* on alternatives.

Another method of building a utility function is to use a weighted average, which is a more general method of specifying a utility function and applies to arbitrary combinatorial domains. A weighted average preference representation [Debreu, 1954] consists of a vector of value functions $\boldsymbol{V} = (v_1, v_2, \ldots, v_m)$ and a vector of weights $\boldsymbol{W} = (w_1, w_2, \ldots, w_m)$. Each function $v_i$ maps the domain of an attribute $i$ onto the natural numbers. The utility of an alternative $\alpha$ is calculated by:

$$u(\alpha) = \sum_{i=1}^{m} w_i * v_i(\alpha_i)$$

Consider a multivalued domain with three attributes $a, b$, and $c$ where each attribute has three possible values, $0, 1$, and $2$. We define a weighted average model by setting the value functions as follows:

$$v_a(x) = 2 - x$$

$$v_b(x) = x$$

$$v_c(x) = x.$$

We specify the weights $(1, 2, 4)$. This means that the alternative $(0, 1, 2)$ creates the sum $1(2) + 2(1) + 4(2) = 12$ and thus the utility of $(0, 1, 2)$ is 12. Using this example we can calculate $u((2, 1, 1)) = 6$ and so $(0, 1, 2) \succ (2, 1, 1)$. This representation is sometimes referred to as a *parts-worth model.*

While all quantitative preference representations convert alternatives into numerical values there are also qualitative representations that do the same, particularly ranking functions. Given a ranking function it is possible, without loss of information, to describe its preorder without the notion of ranks. Thus, ranking functions describe qualitative, not quantitative, preferences despite converting alternative into numerical values before comparison. One method of building a ranking function is to modify the preference formula representation by assigning ranks to each formula in a set of formulas, $\boldsymbol{\varphi}$.

As we noted a limitation of the language preference formulas is that only very simple preorders can be represented. One method of addressing this limitation is to consider a set of formulas, arranged according to their importance to form a preference representation, we call such a representation a *ranking preference formula.*

A *ranking preference formula* (RPF) over the binary combinatorial domain $\mathcal{C}(\mathcal{V})$ consists of a vector of preference formulas $\boldsymbol{\varphi} = (\varphi_1, \varphi_2, \ldots, \varphi_k)$, where $\varphi_i \in \mathcal{L}(\mathcal{V})$, for $i = 1, \ldots, n$. We define the *rank* of an alternative $\alpha$, $r(\alpha)$ by setting $r(\alpha) = \min\{i : \alpha \models \varphi_i\}$. Given two alternatives $\alpha, \beta \in \mathcal{C}(\mathcal{V})$, we define $\alpha \succeq \beta$ if $r(\alpha) \leq r(\beta)$. We note that if $\{i : \alpha \models \varphi_i\} = \emptyset$, $r(\alpha) = \infty$. Thus, alternatives which do not satisfy any of the formulas in $\boldsymbol{\varphi}$ are less preferred than any other alternative. Ranking preference formulas always generate total preorders.

A *preference theory* (PT) $P$ consists of a set of RPFs. For each formula in $P$ an alternative $\alpha$ is assigned a rank, or satisfaction value. These satisfaction values are collected into a *satisfaction vector* $S(\alpha)$ where each entry in the satisfaction vector is associated with a specific RPF (according to some specified order of the

RPFs in $P$), thus $S(\alpha)_1$ refers to the satisfaction value for the first formula and so on. Given a preference theory $P$ and two alternatives $\alpha, \beta$ we define the preference relation determined by $P$, written $\succeq_P$, by setting $\alpha \succeq_P \beta$, if for every RPF $c \in P$, $\text{rank}_c(\alpha) \leq \text{rank}_c(\beta)$. Preference theories are related to the ASO preference language [Brewka et al., 2003]. The original ASO formalism by Brewka et al. [2003] also allows for the specification of hard constraints which limit the preference relation to a subset of the related combinatorial domain consisting of all alternatives satisfying those constraints. PTs ignore this aspect of ASOs.

The preference theory $P$ can be further extended by assigning ranks to its formulas. Given a ranked preference theory $P$, the preference relation between $\alpha$ and $\beta$ is determined by first comparing the alternatives on the lowest ranked (most important) formulas. Any result of this comparison other than equivalence is taken as the result of the comparison under the entire PT. If the two alternatives are equivalent wrt the set of the lowest ranked rules we move on to the next lowest ranked group and proceed in the same way. If the alternatives are equivalent wrt each group of rules considered, they are equivalent wrt $P$. This addition of ranks to $P$ can be viewed as a type of meta-preference, i.e. preferences over preferences.

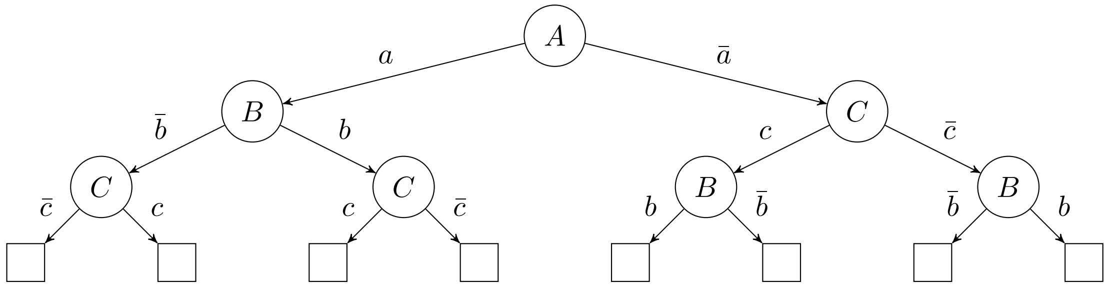
Preference formulas and their generalizations are not the only method of providing a qualitative preference representation. Another important concept underlying several preference representation formalisms is that of lexicographic ordering. A *lexicographic preference model* (LPM) [Fishburn, 1974] represents preferences over a combinatorial domain by assigning an importance order to the attributes and a preference order for each attribute domain. In an LPM an alternative $\alpha$ is preferred to another alternative $\alpha'$ if $\alpha$ has a more preferred value on the most important attribute where it differs from $\alpha'$. Formally, an LPM is a pair $\pi = (r, \succ)$, where $r$ is a strict ranking (a one-to-one mapping of attributes to integers) of attributes and $\succ = \{\succ_{v_1}, \succ_{v_2}, \ldots, \succ_{v_n}\}$ is a collection of total orders, with one order for each attribute's domain. Preference is formally defined as $\alpha \succ_\pi \beta$ if there exists some attribute $a$ such that $\alpha[a] \succ_a \beta[a]$ and for all attributes $b$ such that $r(b) < r(a)$, $\alpha[b] = \beta[b]$.

Again, consider the binary domain defined over $O = \{a, b, c, d\}$. Suppose we have an LPM $\pi = (r, \succ)$ where the ranking $r$ induces the order $d \succ c \succ b \succ a$ on attributes and we prefer the inclusion of an object over its exclusion. In this setting the alternatives $\{a, b, d\}$ and $\{a, c\}$ are ordered such that $\{a, b, d\} \succ \{a, c\}$. The reason that they are ordered this way is that the most important attribute where the two alternatives differ is $d$, and the inclusion of $d$ is preferred over its exclusion. The order represented by $\pi$ can be replicated using a weighted average model where the value function gives inclusion the value 1 and exclusion the value 0 for each attribute in the domain. If we then set the weights to $(1, 2, 4, 32)$ the resultant weighted average model is equivalent to $\pi$. While the induced orders are identical between the two models the information that $d$ is heavily weighted for cannot be explicitly extracted from the LPM. This shows that quantitative representations may induce the same orders as qualitative representations and can provide additional information.

LPMs represent preferences in an unconditional manner, that is preferences do not change based on properties of the alternatives being compared. *Lexicographic preference trees* (LP-trees) introduced by Booth et al. [2010] generalize LPMs to allow them to express conditional preferences.

An LP-tree modifies an LPM by replacing the linear order of attributes with a tree. Each vertex in the tree is labeled with a singular attribute $v_i$ and has a number of descendants equal to the size of its domain, $|\mathcal{D}(v_i)|$. LP-trees have the additional condition that each path from the root to a leaf has exactly one occurrence of each attribute as a label. Like LPMs each vertex has an associated linear order over the domain of its attribute, different vertices with the same associated attribute may also have different linear orders. The process of determining dominance between two

Figure 2.3: An LP-tree



alternatives begins at the root of the tree. As long as both alternatives are equal, with respect to the attribute labeling the current vertex, one follows the edge labeled with the shared value of that attribute. Once one reaches a vertex where the alternatives differ the alternative with the more preferred value for that attribute is preferred over the other alternative. If one reaches a leaf then the alternatives are equally preferred (in fact, they are the same).

Figure 2.3 shows an example of an LP-tree over a binary combinatorial domain. It is important to note that an arbitrary LP-tree is exponential in size wrt to the number of attributes. In order for LP-trees to be compact we must place restrictions on their form. Common restrictions include removing conditional importance (each path through the tree orders the attributes the same) and/or removing conditional preference (the preference order over an attribute's domain does not change based on its location in the tree). It is easy to see that an LP-tree without either conditional importance or conditional preference is equivalent to an LPM.

A limitation of LPMs and LP-trees is that they only consider attributes one at a time. The process of considering several attributes at a time is called *grouping* [Wilson, 2009]. The preference language of *conditional lexicographic preference models* (CLPMs) [Huelsman and Truszczyński, 2020] incorporates grouping into LP-trees with unconditional attribute importance. A CLPM is a pair $\pi = (r, T)$, where $r$ is an importance ranking of attributes and $T$ is a set of conditional preference tables

(CPTs), one for each attribute.

Given an attribute $v \in \mathcal{V}$ which is conditioned by a set of attributes $P_v$ the CPT associated with $v$ contains one row for each possible selection of values for attributes in $P_v$. Each row specifies a linear order on the values in the domain of $v$, the relation induced by the CPT entry is denoted $\succeq_{T,v}$. An additional restriction is placed on the ranking function $r$ such that for a pair of attributes $a, b \in \mathcal{V}$ if $a$ conditions $b$, that is changes in $a$ alter preferences over the value of $b$, then $r(a) < r(b)$. Like a regular LPM a CLPM considers attributes in order of importance according to $r$, if $r$ induces a total order over $\mathcal{V}$ then the CLPM is a compact representation of a certain class of LP-tree. The preference relation defined by a CLPM is as follows:

**Definition 1.** *Let $\pi = (r, T)$ be a CLPM over a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$. We define the relation $\succeq_\pi$ as follows. For any two alternatives $\alpha, \beta \in \mathcal{C}(\mathcal{V}, \mathcal{D})$, $\alpha \succeq_\pi \beta$ if $\alpha = \beta$, or for some attribute $a \in \mathcal{V}$ we have $\alpha[a] \succ_{T,a} \beta[a]$, for all $b \in \mathcal{V}$ such that $r(b) \leq r(a)$ we have $\alpha[b] \succeq_{T,b} \beta[b]$ and for all $c \in \mathcal{V}$ such that $r(c) < r(a)$ we have $\alpha[c] = \beta[c]$.*

This concept is well defined, the additional restriction on $r$ and $T$ allows us to perform the required attribute comparisons by ensuring that any conditioning attributes will be equal in both alternatives, before the attribute comparison has to be made. By relaxing the importance ranking from a strict total order to a total preorder we have created the possibility of disagreement among equally important attributes. In such cases neither $\alpha \succ_\pi \beta$ nor $\beta \succ_\pi \alpha$ holds, meaning the alternatives are incomparable. We denote this as $\alpha \bowtie_\pi \beta$.

Consider the binary domain we defined previously over $O = \{a, b, c, d\}$. An example of a CLPM over that domain is:

$$r = a \succ \{b, c\} \succ d$$

with our CPTs being those in Table 2.1.

Table 2.1: Example conditional preference tables.

| $a$ |
|---|
| $a \succ \bar{a}$ |

| $b$ |
|---|
| $b \succ \bar{b}$ |

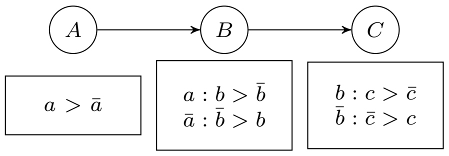
| $c$ | |
|---|---|
| $a$: | $c \succ \bar{c}$ |
| $\bar{a}$: | $\bar{c} \succ c$ |

| $d$ | |
|---|---|
| $a \wedge c$: | $d \succ \bar{d}$ |
| $a \wedge \bar{c}$: | $d \succ \bar{d}$ |
| $\bar{a} \wedge c$: | $\bar{d} \succ d$ |
| $\bar{a} \wedge \bar{c}$: | $d \succ \bar{d}$ |

In this example we have $\{b, c, d\} \succ \{b, c\}$ since the first rank where the two differ in value is the last rank and according to $T$, $\bar{d} \succ d$ if $\bar{a} \wedge c$. It is important to note that although $b$ could be conditioned on the value of $a$ it does not need to be, and in the example is not. One of the factors which differentiates CLPMs from LPMs and LP-trees is that a CLPM can have pairs of alternatives that are incomparable. Using our example $\{a, b, d\}$ is incomparable with $\{a, c\}$, $\{a, b, d\} \bowtie \{a, c\}$. This incomparability occurs because in the second rank the conditional preference table says that $b \succ \bar{b}$ and $c \succ \bar{c}$ given that $a$ is present in both alternatives. This means that according to the CPT for $b$, $\alpha$ is preferred and according to the CPT for $c$, $\beta$ is preferred. This prevents us from deriving a preference relation in either direction meaning $\alpha \bowtie \beta$.

A common graphical method of representing conditional preferences is offered by *conditional preference networks* (CP-nets). A CP-net for a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ is a pair $(G = (\mathcal{V}, E), T)$ where $G$ is a digraph with $\mathcal{V}$ as the set of vertices, identical to the set of attributes, $E$ is the set of edges in $G$, and $T$ is the set of CPTs, one for each vertex. A directed edge between two vertices $(u, v)$ in $G$ indicates that attribute $v$ is conditioned on attribute $u$. CPTs are defined identically as above. The preference relation defined by a CP-net is the transitive closure of the local preference relation $\succ_T$ specified by the CPTs. Given two alternatives $\alpha$ and $\beta$, that differ on a single attribute $v_i$, we define $\alpha \succ_T \beta$ if $\alpha[v_i] \succ \beta[v_i]$ according to $v_i$'s CPT. Each pair of alternatives $(\alpha, \beta)$ such that $\alpha \succ_T \beta$ ($\beta \succ_T \alpha$, respectively) is called a *worsening flip* (*improving flip*, respectively). The preference relation specified by a given CP-net $C$, $\succeq_C$, is defined by setting $\alpha \succeq_C \beta$ if $\alpha = \beta$ or there exists a series of alternatives

Figure 2.4: A CP-Net



$\gamma_1, \gamma_2, \ldots, \gamma_k$ such that $\gamma_i \succeq_T \gamma_{i+1}$ for $1 \leq i < k$ and $\alpha = \gamma_1$ and $\beta = \gamma_k$. The sequence of alternatives used to show a preference relation between alternatives in a CP-net is known as a *flipping sequence*. If no flipping sequence exists between two alternatives then they are considered *incomparable*, denoted $\bowtie_C$.

Using the example CP-net in Figure 2.4 to compare the alternatives $(a, b, c)$ and $(\bar{a}, b, \bar{c})$ we look for a sequence of worsening flips from $(a, b, c)$ to $(\bar{a}, b, \bar{c})$. One such sequence is $(a, b, c) \succ (\bar{a}, b, c) \succ (\bar{a}, b, \bar{c})$ and so we know that $(a, b, c) \succ (a, \bar{b}, \bar{c})$ according to the CP-net in Figure 2.4.

Both the CP-net and the CLPM preference representations use CPTs to express conditional preference information. While the two representations are similar [Huelsman and Truszczyński, 2019] they interpret conditional preferences differently, CLPMs lack the requirement that "all else being equal", which means a CLPM and CP-net with the same CPTs may generate different preference orders.

## 2.4 Preference Reasoning, Learning, and Aggregation

Now that we have a common vocabulary to talk about how preferences are represented it is important to know what types of reasoning we might wish to accomplish using preferences. Given a domain $\Omega$ of alternatives and a preference relation $\succeq$ on $\Omega$, the key reasoning tasks are:

- Dominance: Given alternatives $\alpha, \beta \in \Omega$, determine whether they are comparable and, if so, which of them is preferred.

- Optimality: Find an alternative that is preferred over any other (or determine that no such alternative exists.)
- Maximality: Find a maximal alternative (an alternative that is not strictly dominated by any other alternative.)
- High-quality: Find an alternative that is strictly dominated by no more than $k$ other alternatives.

It is the focus of preference reasoning to automate these tasks through the development of efficient algorithms. Unfortunately, the computational complexity of such algorithms depends on the preference representation language used. We define the dominance task below:

**Problem 1** (Dominance). *Given a model $M$ over a domain $\Omega$ and two alternatives $\alpha, \beta \in \Omega$, does $\alpha \succ_M \beta$ hold?*

The dominance problem defines the task of deriving the strict preference relation of a model. A particular instance of the dominance problem is called a dominance query. Solving a dominance query takes polynomial time, with respect to the number of attributes, for LPMs [Kohli and Jedidi, 2007] but is NP-hard for acyclic CP-nets [Boutilier et al., 2004] and PSPACE-complete for arbitrary CP-nets [Goldsmith et al., 2008].

Sometimes it may be useful to simply know which alternative is best, or most preferred. There are two distinct ways to answer such questions: optimality and maximality.

**Problem 2** (Optimality). *Given a model $M$ over a domain $\Omega$ and an alternative $\alpha \in \Omega$, does $\alpha \succ_M \beta$ hold for all $\beta \in \Omega$?*

**Problem 3** (Maximality). *Given a model $M$ over a domain $\Omega$ and an alternative $\alpha \in \Omega$, does $\beta \not\succ_M \alpha$ hold for all $\beta \in \Omega$?*

Problem 2 defines whether an alternative is as good as any other alternative. This problem defines a stronger property than Problem 3 which looks instead for maximally preferred alternatives, which allows for the possibility of multiple incomparable best alternatives.

These reasoning problems extract information from preference models, but another related problem is how to build preference models we reason about. An agent might be able to specify them directly; or one could collect a set of comparisons made by the agent and use them to construct a model of a particular preference language which best fits these observations (preference learning). The problem of learning preferences has two variants: active and passive. In active learning we ask questions to the agent whose preferences we wish to model. These questions consist of giving the agent a pair of alternatives $(\alpha, \beta)$ and the agent returns the alternatives ordered according to their preferences. In active learning the goal is to construct a good model of the agent's preferences while minimizing the number of queries. The process of active learning comes coupled with the question of how many queries are needed to learn a model of a particular language. This problem is known as the *communication complexity* of a language. The communication complexity of LP-trees has been studied and it was found that the addition of conditional preferences made the communication complexity intractable [Bräuning and Hüllermeier, 2016]. Our work is not concerned with active learning and focuses solely on passive learning.

Passive learning starts with a static set of pairwise comparison examples. Each example consists of an ordered pair of alternatives $(\alpha, \beta)$ taken to mean that $\alpha \succ \beta$ (the agent strictly prefers $\alpha$ to $\beta$). The notion of an example can be further generalized to a triple $(\alpha, \beta, R)$ stating the relation between $\alpha$ and $\beta$. For instance, $(\alpha, \beta, \bowtie)$ states that $\alpha$ and $\beta$ are incomparable according to the agent. In passive learning the goal is to construct a model from a selected language that best fits the available examples. In general, passive learning makes no guarantee that the learned model will fully agree with the example set. Learning algorithms are typically designed to minimize disagreement between the model and the example set, i.e. finding the best possible model. In the case of LPMs the problem of finding a model which agrees with an example set, assuming such a model exists, takes polynomial time. The problem of finding the model which minimizes disagreement, when no LPM model completely agrees with the example set is NP-complete [Schmitt and Martignon, 2006].

Reasoning about the preferences of a single agent is important, but in many settings we are interested in the collective preferences of a group of agents, where each agent has their own unique preferences. The process of combining the preferences of multiple agents is referred to as *preference aggregation*. Preference aggregation normally takes the form of voting [Rothe, 2016]. Voting is a method of converting a set of preferences (votes) over a number of objects (candidates) into a set of winners.

Voting over combinatorial domains is often difficult because we have implicit representations of orders (votes) not the orders themselves. For example, if we are voting with plurality we need to know the most preferred alternative for each voter. If all agents are represented by LP-trees the task of finding each agent's optimal alternative and the plurality winner is easy to compute [Booth et al., 2010]. If we decide to vote using the Borda voting rule instead the problem becomes NP-hard [Lang et al., 2018] (assuming conditional preferences). There has been work on implementing voting schemes using compact preference representations. Despite clever implementations the complexity of voting using many different compact preference languages varies. In some cases it is possible to vote in polynomial time [Liu and Truszczyński, 2019], typically with some restriction [Lang and Xia, 2009], but in other cases it may be computationally difficult [Conitzer et al., 2011].

While finding winners according to a voting scheme will produce a singular decision, we may want to make group decisions in the future, and voting each time can become tedious. One way to aggregate the preferences of multiple agents, without voting each time, is to learn a *joint preference model.* A joint preference model is a preference model which represents the preferences of a group, rather than a singular agent. Joint preference models can be built in any preference language and the process is very similar to learning a preference model for a single agent, except that examples used for learning come from every agent in the group. This brings up the problem that the learned preference model be *fair*.

There is no single definition of fairness. An aggregation is considered fair when it has "good" properties with respect to the agents who make up the group. To introduce fairness criteria formally, we will need to refer to the examples of a particular agent $a$ in a given example set $\varepsilon$. We will denote this set of examples as $\varepsilon(a)$. A common notion of fairness is the utilitarian criterion [Rothe, 2016]. According to the utilitarian criterion the best joint preference model is the one which satisfies the largest number of examples. Formally, a model $M$ over a set of examples $\varepsilon$ obtained from a set of agents $A = \{a_1, \ldots, a_k\}$ is given the utilitarian score:

$$s_u(\succeq, \varepsilon, M) = \sum_{a \in A} |\{e|e \in \varepsilon(a) : M \models e\}|$$

.

The utilitarian criterion can be contrasted with the maximin criterion. Under maximin the joint preference model which satisfies the largest number of examples of the agent with the least number of satisfied examples is the best model. This approach guarantees that the agent with the least number of satisfied examples has as many examples satisfied as possible, thus guaranteeing a minimal level of satisfaction for each agent. Formally, a model $M$ over a set of examples $\varepsilon$, over a set of agents $A = \{a_1, \ldots, a_k\}$, is given the maximin score:

$$s_{mm}(\succeq, \varepsilon, M) = \min_{a \in A} |\{e|e \in \varepsilon(a) : M \models e\}|$$

The goal of learning a joint preference model is to maximize its fairness score, in this setting that means either the utilitarian or maxmin score. To illustrate the difference between utilitarian and maximin aggregations consider two agents, $A$ and $B$, who have preferences over a domain $\Omega = \{a, b, c, d\}$. Suppose agent $A$ has a preference order given by $a \succ_A b \succ_A c \succ_A d$ and $B$ by $d \succ_B c \succ_B b \succ_B a$ and our full example set consists of all possible strict preference comparisons from each agent, totaling six examples from each agent. Now consider a potential joint order $J_1$ such that $a \succ_{J_1} b \succ_{J_1} c \succ_{J_1} d$. In this setting six examples are satisfied for agent $A$ and none for agent $B$ giving $J_1$ a utilitarian score of 6 and a maximin score of 0. These scores are identical if we choose a joint order $J_2$ such that $d \succ_{J_2} c \succ_{J_2} b \succ_{J_2} a$, where six examples are satisfied for agent $B$ and none for agent $A$. Finally, consider the joint order $J_3$ where $a \succ_{J_3} d \succ_{J_3} c \succ_{J_3} b$. In this setting three examples are satisfied for agent $A$ and three examples are satisfied for agent $B$, meaning $J_3$ has a utilitarian score of 6 and a maximin score of 3. According to the utilitarian score $J_1, J_2$, and $J_3$ are all identically performing aggregations. On the other hand, according to the maximin score, $J_3$ is the best joint preference order.

As we can see, $J_3$ blends the two agent's orders together to produce a joint preference model and so one can argue that this is a fairer method of aggregating the agents' preferences. While maximin may be considered fairer, it can be hard to compute [Bartholdi et al., 1989], even when we only use it as a voting scheme.

When we learn preferences from a single agent there is an implicit assumption that the examples we use do not contradict each other. In the multi-agent setting we cannot make this assumption as it is common for agents to disagree. Thus, multi-agent example sets are likely to have conflicts. The normal manner of learning a preference model is codified as follows:

**Definition 2** (Learning a Preference Model)**.** *Given a set of example $\varepsilon$ and a preference language $\mathcal{L}$ find the model $M \in \mathcal{L}$ such that $M$ satisfies as many examples in*

*$\varepsilon$ as possible.*

If we allow for example sets with conflicting examples then this problem subsumes utilitarian based learning for multi-agent settings. Indeed, utilitarian criterion focuses only on how many examples were satisfied in total; i.e. which agent an example comes from does not affect the utilitarian score. Simply put, if we strip all the agent information from an example set and build a model that maximizes the number of satisfied examples then that model also maximizes the utilitarian score.

When learning a preference model the language used is commonly selected a priori without considering any other factors. Therefore, it is possible that the language is a poor matchup for representing the example set, that is, it does not contain models that would closely match the examples. The solution to this problem is to select a possible rich (expressive) preference representation language or to select the preference representation language only after the example set is given.

The second approach introduces the following problem:

**Definition 3** (Preference Representation Language Selection)**.** *Given a set of examples $\varepsilon$ and a set of preference representation languages $\boldsymbol{\mathcal{L}} = \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_k\}$ select the language $\mathcal{L} \in \boldsymbol{\mathcal{L}}$ such that learning a model $M \in \mathcal{L}$ will maximize the number of examples satisfied in $\varepsilon$.*

Perfectly solving the language selection problem would allow us to ensure that we select the best language to learn in any situation. While a perfect solution may not be possible a sufficient approximation may allow for an increase in learning accuracy.

# Chapter 3 Relating Preference Languages By Their Expressive Power

We define a language's expressive power in terms of the class of preorders its models can induce. This allows us to compare languages by means of the inclusion relation which, in this context, we refer to as *preference representation language subsumption* or, simply, *subsumption*. The subsumption relation, because it is essentially set-inclusion, yields a partial preorder over any set of preference representation languages.

## 3.1 Theoretical Framework

Preference languages consist of a syntax and a semantics. Each particular syntactically valid combination of preference statements defines a model of that language. Interpreting a model based on the language semantics gives us a preference ordering. Since models directly imply a preference order we will use models as a shorthand to mean a preference ordering. Since subsumption relies on equivalence between expressed preferences we must first define what we mean when we say two models express the same preference ordering.

**Definition 4** (Preference Model Equivalence). *Given two models $M$ and $M'$ selected from some two preference representation languages over the same domain of alternatives $\Omega$, $M$ is equivalent to $M'$, $M' \equiv M$, if for all pairs $\alpha, \beta \in \Omega$, $M \models \alpha \succeq \beta$ if and only if $M' \models \alpha \succeq \beta$.*

Using this definition of model equivalence we define our notion of subsumption as follows:

**Definition 5** (Preference Language Subsumption). *Given two preference languages $L$ and $L'$, $L$ subsumes $L'$, $L' \subseteq L$, if for every model $M' \in L'$ there exists a model $M \in L$ such that $M' \equiv M$.*

This definition of subsumption means that a subsumption relation exists between two languages if a model in one language can be transformed into an equivalent model of the other language. This allows us to prove a subsumption relation exists by showing a mapping from models of one language to models of another. These mappings are similar to mapping reductions used to prove computational hardness, although these mappings do not need to be polynomial time algorithms. If a polynomial time mapping can be constructed then we call the corresponding subsumption relation *efficient*.

**Definition 6** (Efficient Subsumption). *Given two preference languages $L$ and $L'$, $L$ efficiently subsumes $L'$ if there exists a polynomial time mapping $f : L' \to L$, such that given a model $M \in L'$, $f(M) \in L$ and $f(M) \equiv M$.*

Efficient subsumption provides insight into the computational complexity of preference reasoning problems in different languages. For example, if $\mathcal{L}$ efficiently subsumes $\mathcal{L}'$ and dominance is NP-hard for $\mathcal{L}'$ then it is also NP-hard for $\mathcal{L}$ since we can construct a polynomial time mapping reduction from the dominance problem in $\mathcal{L}'$ to the dominance problem in $\mathcal{L}$. While efficient subsumption is of interest we do not pursue it in this work.

Subsumption allows us to define *language equivalence* relations. Simply put two languages are equivalent when they subsume each other. We state this definition formally below:

**Definition 7** (Preference Language Equivalence). *Given two preference languages $L$ and $L'$, $L$ is equivalent to $L'$, $L \sim L'$ if $L \subseteq L'$ and $L' \subseteq L$.*

Subsumption and its associated relations allow us to compare and group languages based on their expressive power. By applying subsumption to a set of languages we can determine the representative language of the set, i.e. the language which subsumes all other languages in the set, if it exists.

**Definition 8** (Representative Language). *A preference language $L$ is a representative preference language of a set of preference languages $\boldsymbol{L}$ if $L \in \boldsymbol{L}$ and for all $L' \in \boldsymbol{L}$, $L' \subseteq L$.*

The representative language of a set can replicate any model from any other language in that set. Of course a set of langauges may contain no representative langauge. This occurs when two langauges in a set are incomparable according to the subsumption relations. For example if a set of languages contains LPMs and CP-nets then that set will not have a representative language because LPMs and CP-nets are incomparable according to subsumption, as we will show later. In these cases it makes sense to find the smallest subset of those languages which retain the full expressive power of the larger set. We formally define the minimal language set of a set of languages below.

**Definition 9** (Minimal Representative Language Set). *Given a set of preference representation languages $\boldsymbol{L}$ the set $\boldsymbol{L'}$ is a minimal representative language set of $\boldsymbol{L}$ if for all $L \in \boldsymbol{L}$ there exists an $L' \in \boldsymbol{L'}$ such that $L \subseteq L'$ and there does not exist a pair of languages $L, L' \in \boldsymbol{L'}$ such that $L \subseteq L'$.*

Ideally, in any application where we might need a set of languages knowing the minimal representative language set would be beneficial, since it would eliminate the need to consider languages whose models could be expressed by another language in the set. Thus, computing the minimal rperesentative language set minimizes the number of languages in the set while maintaining the set's expressive power (as expressed by the class of preorders models of its member languages may produce.)

## 3.2 Subsumption Proof Techniques

The core strategy when proving a subsumption relation is to build a mapping from models of one language to equivalent models in another language. While subsumption could be used to compare two languages over a concrete domain of specific alternatives, we will deal with it specifically over the class of all possible combinatorial domains.

Proving a lack of subsumption between two languages can be more difficult. The most direct method of proving that two languages are not related by subsumption is to show that one of the languages can express a specific preference ordering the other cannot. Moreover this particular approach means that the proof is specifically tuned to the eccentricities of that particular language and so may not be readily generalized to other situations.

An easy way to demonstrate that one language can induce a preference ordering that another cannot is to show that a particular derived relation between some two alternatives holds in some model in the first language, but cannot not hold in any model from the second language. For instance, if two alternatives are incomparable in some model from the first language and are comparable in all models from the second language then the second language does not subsume the first language. We formally state this below in Lemma 1.

**Lemma 1.** *Given two languages $\mathcal{L}$ and $\mathcal{L}'$ and a preference relation $R \in \{\succ, \approx, \bowtie\}$ if there exists a model $M \in \mathcal{L}$ over a domain $\Omega$ such that $\alpha R_M \beta$ for two alternatives $\alpha, \beta \in \Omega$ and there exists no model $M' \in \mathcal{L}'$ such that $\alpha' R_{M'} \beta'$ for two alternatives $\alpha, \beta \in \Omega$ then $\mathcal{L} \not\subseteq \mathcal{L}'$.*

*Proof.* Each possible relation $R$ requires a specific structure of $\succeq$ relations between alternatives. This means that if it is possible for $R$ to be derived between two alternatives based on the induced $\succeq$ relation of some model $M \in \mathcal{L}$ and no model in $\mathcal{L}'$ can derive $R$ from its induced $\succeq$ relation, then there cannot exist a model $M' \in \mathcal{L}'$ which is equivalent to $M$, therefore $\mathcal{L}$ is not subsumed by $\mathcal{L}'$. $\square$

Another way of proving or disproving subsumption is to exploit transitivity.

**Lemma 2.** *Given three languages $\mathcal{L}$, $\mathcal{L}'$, and $\mathcal{L}''$ if $\mathcal{L}'' \subseteq \mathcal{L}'$ and $\mathcal{L}' \subseteq \mathcal{L}$ then $\mathcal{L}'' \subseteq \mathcal{L}$.*

**Lemma 3.** *Given three languages $\mathcal{L}$, $\mathcal{L}'$, and $\mathcal{L}''$ if $\mathcal{L}' \subseteq \mathcal{L}$ and $\mathcal{L}'' \not\subseteq \mathcal{L}$ then $\mathcal{L}'' \not\subseteq \mathcal{L}'$.*

**Lemma 4.** *Given three languages $\mathcal{L}$, $\mathcal{L}'$, and $\mathcal{L}''$ if $\mathcal{L}' \subset \mathcal{L}$ and $\mathcal{L}'' \subseteq \mathcal{L}'$ then $\mathcal{L} \not\subseteq \mathcal{L}''$.*

Some properties of preference languages narrow down the class of preorders that can be induced by their models and, in this way, can assist in establishing subsumption. A simple example of this is that every preorder defined by a utility or ranking function is a total preorder (Lemma 5).

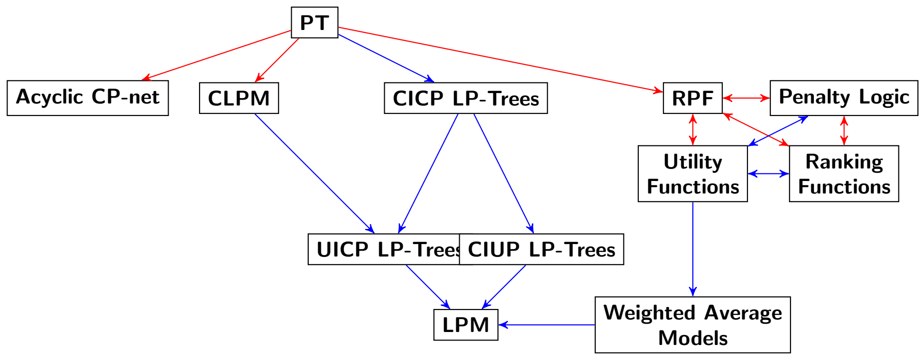
**Lemma 5.** *If every model $M$ in a preference language defines its preorder by means of some utility function $f_M$ and one of the relations $\geq$ or $\leq$ on reals (so that we have $M \models \alpha \succeq \beta$ if and only if $f_M(\alpha) \leq f_M(\beta)$, or in the other case $f_M(\alpha) \geq f_M(\beta)$), then every preference order defined by a model from that language is a total preorder.*

While these results are trivial they simplify the process of establishing subsumption for some collections of preference languages. Moreover such results allow us to derive some subsumption relations from other subsumption relations. Thus, it is not always necessary to compare each pair of languages in a set, but instead focus on a smaller subset of pairs.

## 3.3 Case Study: Combinatorial Domains

We now consider preference languages for representing preference orders on a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ and establish the subsumption relations between them. The results are summarized in Figure 3.1. Each vertex in the graph represents a preference

Figure 3.1: Each edge $(u, v)$ means that preference language $u$ subsumes $v$. Blue edges are known from the literature.



representation language with an edge between two vertices indicating subsumption, the origin vertex subsumes the destination vertex. If no edge is present between two vertices and there is no path between them then the languages are incomparable.

The graph in Figure 3.1 represents both information from the literature as well as our own work. From the literature we learn that CICP LP-trees subsume UICP and CIUP LP-trees since they represent specializations of CICP LP-trees [Booth et al., 2010]. LPMs (which are UIUP LP-trees) are subsumed by both UICP and CIUP LP-trees, PTs subsume RPFs [Brewka et al., 2003], utility functions subsume weighted average models, and CLPMs subsume UICP LP-trees [Huelsman and Truszczyński, 2020], since the former languages represent generalizations of the later languages. The literature also shows us that weighted average models subsume LPMs [Kohli and Jedidi, 2007] and that PTs can model CICP LP-trees [Zhu, 2016].

As a first step to proving the relations in Figure 3.1 we look at a cluster of languages (RPFs, ranking functions, penalty logic, and utility functions) which are equivalent to each other. This proof begins by showing that a ranking function can replicate any given total preorder over a finite domain, see Lemma 6.

**Lemma 6.** *Given a total preorder $\succeq$ over a finite domain $\Omega$ there exists a ranking function $r$ such that the preference relation induced by $r$, $\succeq_r$, is equivalent to $\succeq$.*

*Proof.* Given an arbitrary total preorder $\succeq$ over a finite domain we first group alternatives that are equally preferred as though they were singular alternatives; we now have a linear order over groups of equally preferred alternatives. Members of the most preferred group of alternatives are assigned rank 1, the members of the next most preferred group rank 2, and so on until all alternatives are given a rank. Thus for two alternatives $\alpha$ and $\beta$ if $\alpha \succeq \beta$ and $\beta \succeq \alpha$ (the two alternatives are equally preferred) $r(\alpha) = r(\beta)$ and if $\alpha \succeq \beta$ and $\beta \not\succeq \alpha$ then $r(\alpha) < r(\beta)$. This means the preference order induced by $r$ is the same as $\succeq$. Therefore, it is possible to replicate any total preorder using a ranking function. □

Using this proof we can show that the cluster containing ranking functions, utility functions, penalty logic, and ranking preference formulas are all equivalent languages, since they induce total preorders and also subsume ranking functions. Moreover, this cluster represents a set of languages which includes any total preorder over any combinatorial domain.

Before we can prove these edges there is one important concept to define. Consider a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ and its related language of propositional logic $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$. Here $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$ contains one atom for each attribute-value pair, rather than each attribute (as occurs in the binary combinatorial domain setting). That is, if an alternative $\alpha$ has the value $j$ for attribute $v_i \in \mathcal{V}$ then the atom $x_{i,j}$ is true. This is an extension of the alphabet we defined earlier for binary combinatorial domains. Similarly, there is a one-to-one correspondence between alternatives in $\mathcal{C}(\mathcal{V}, \mathcal{D})$ and truth assignments to atomic formulas in the language $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$. This means that it is possible to construct a formula $\phi$ using $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$ such that $\phi$ is only satisfied by a given alternative, $\alpha$. We refer to such a formula as $\phi_\alpha$.

**Lemma 7.** *Ranking Function* $\subseteq$ *Penalty Logic*

*Proof.* Consider an arbitrary ranking function $r$ over a domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ which defines a model $M' \in$ Ranking Function. We construct a model $M \in$ Penalty Logic such that for each alternative $\alpha \in \mathcal{C}(\mathcal{V}, \mathcal{D})$, $\phi_\alpha \in M$. Further, we define the weight $w_\alpha$ for $\phi_\alpha$ to be:

$$w_\alpha = b - r(\alpha),$$

where $b$ is the highest rank of any alternative according to $r$. Since $\phi_\alpha$ is the only formula in $M$ satisfied by $\alpha$ its utility is:

$$\begin{aligned} u(\alpha) &= - \sum_{\beta \in \mathcal{C}(\mathcal{V},\mathcal{D}), \beta \neq \alpha} w_\beta \\ u(\alpha) &= w_\alpha - \sum_{\beta \in \mathcal{C}(\mathcal{V},\mathcal{D})} w_\beta \\ u(\alpha) &= b - r(\alpha) - \left( \sum_{\beta \in \mathcal{C}(\mathcal{V},\mathcal{D})} w_\beta \right) \end{aligned}$$

For simplicity we assign the constant $c$ the value of $\sum_{\beta \in \mathcal{C}(\mathcal{V},\mathcal{D})} w_\beta$, which is the sum of all weights in $M$.

*(If $\alpha \succeq_r \beta$ then $\alpha \succeq_M \beta$.)* If $\alpha \succeq_r \beta$ then $r(\alpha) \leq r(\beta)$. The utility of $\alpha$ and $\beta$ in $M'$ compare as follows:

$$\begin{gathered} u(\alpha)?u(\beta) \\ b - r(\alpha) - C?b - r(\beta) - C \\ -r(\alpha) \geq -r(\beta). \end{gathered}$$

This means $u(\alpha) \geq u(\beta)$ and thus $\alpha \succeq_M \beta$.

*(If $\alpha \succeq_M \beta$ then $\alpha \succeq_r \beta$.)* We know that $u(\alpha) \geq u(\beta)$ and so:

$$\begin{gathered} u(\alpha) \geq u(\beta) \\ b - r(\alpha) - C \geq b - r(\beta) - C \\ -r(\alpha) \geq -r(\beta) \\ r(\alpha) \leq r(\beta) \end{gathered}$$

The preference order induced by $M$ is identical to that induced by $M'$, meaning any model in the Ranking Function language can be expressed by a model in Penalty Logic. $\square$

**Lemma 8.** *Ranking Function $\subseteq$ Utility Function*

*Proof.* Consider an arbitrary ranking function $r$ over a domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$. We construct a utility function $u$ such that for any given alternative $\alpha$, $u(\alpha) = -r(\alpha)$. It is evident that higher utility alternatives will have lower ranks and vice versa, thus $r$ and $u$ both induce the same preference relation and any model in the Ranking Function language can be converted into a model in the Utility Function language. $\square$

**Lemma 9.** *Ranking Function $\subseteq$ Ranking Preference Formula*

*Proof.* Consider an arbitrary ranking function $r$ over a domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$. For each rank $k \in r$ we define a boolean formula $\Phi_k$ such that

$$\Phi_k = \vee_{\alpha \in \mathcal{C}(\mathcal{V},\mathcal{D}), r(\alpha)=k} \ \phi_\alpha.$$

We then define an RPF $\boldsymbol{\varphi} = (\Phi_1, \ldots, \Phi_n)$ where $n$ is the highest rank according to $r$. This means for a particular alternative $\alpha$ its rank according to $\boldsymbol{\varphi}$ is the same as its rank according to $r$, thus they induce the same preference relation. This means that all models in the Ranking Function language have an equivalent model in the Ranking Preference Formula language. $\square$

The subsumption relations proved above show that utility functions, RPFs, penalty logic, and ranking functions are all equivalent languages and capable of expressing any possible total preorder over combinatorial domains. Similar to how ranking functions include any total preorder, PTs include any preorder. This means that PTs subsume all the languages we consider here. We prove this result in several steps.

First, we show that if there exists a set of satisfaction vectors associated with a set of alternatives then there exists a preference theory $P$ which induces those satisfaction vectors for those alternatives, see Lemma 10.

**Lemma 10.** *Given a set of alternatives $A = \{\alpha_1, \ldots, \alpha_n\}$ from a combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ and a set of satisfaction vectors $S = \{s(\alpha_1), \ldots, s(\alpha_n)\}$ there exists a preference thoery $P$ which recreates $S$ given $A$.*

*Proof.* Given $A$ and $S$ we construct a preference theory $P = (\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_k)$ such that $\boldsymbol{\varphi}_i$ corresponds to the $i^{\text{th}}$ satisfaction term in the satisfaction vector of $P$, where $k$ is the size of the satisfaction vectors in $S$. We construct each conditional ranking preference formula $\boldsymbol{\varphi}_i \in P$ such that

$$\boldsymbol{\varphi}_i = (\varphi_{i,1}, \ldots, \varphi_{i,m}),$$

where $m$ is the largest value for the $i^{\text{th}}$ value of a satisfaction vector in $S$. Each $\varphi_{i,j}$ is defined as

$$\varphi_{i,j} = \vee_{\alpha \in A, s(\alpha)_i = j}\ \phi_\alpha.$$

In the event that there are rules with empty ranks we assign those formulas to simply be false, meaning they are never satisfied. Additionally, all conditional formulas for each ranking preference formula are set to true, meaning they are always satisfied. It is simple to see that given the construction of $P$ any alternative $\alpha \in A$ will induce $s(\alpha)$ as its satisfaction vector. □

Lemma 10 implies that if one could construct a set of satisfaction vectors which induce the relations of a particular preorder $\succeq$ then there is a preference theory which induces that preorder. Part of constructing such a set of satisfaction vectors is describing a canonical ranking for any arbitrary preorder. The canonical ranking for a preorder $\succeq$ describes a total preorder $\succeq'$ such that if $\alpha \succeq \beta$ then $\alpha \succeq' \beta$. Simply put the canonical ranking converts a preorder into a total preorder by adding relations between pairs of alternatives that are incomparable in the original preorder.

**Definition 10** (Canonical Ranking Function)**.** *The canonical ranking function of a given preference relation $\succeq$, $r_{c,\succeq}$ is defined as follows:*

$$r_{c,\succeq}(\alpha) = \begin{cases} 1 & |Dom(\alpha)| = 0 \\ 1 + \max_{\beta \in Dom(\alpha)} r_{c,\succeq}(\beta) & |Dom(\alpha)| > 0 \end{cases}$$

*Where $Dom(\alpha)$ is the set of all alternatives in $\beta \in \Omega$ such that $\beta \succ \alpha$.*

Given a canonical preorder (induced by the canonical ranking function) we can extract some useful information about the original preference relation between pairs of alternatives which will help when it comes to constructing satisfaction vectors, see Lemmas 11 and 12.

**Lemma 11.** *Given a partial preorder $\succeq$ if $r_{c,\succeq}(\alpha) = r_{c,\succeq}(\beta)$ then either $\alpha \approx \beta$ or $\alpha \bowtie \beta$.*

*Proof.* Suppose not. Since the cases are symmetrical we will handle the case where $\alpha \succ \beta$ and $r_{c,\succeq}(\alpha) = r_{c,\succeq}(\beta)$. In this case we know, by Definition 10, that either $\alpha$ and $\beta$ are undominated, which cannot be the case since $\alpha \succ \beta$, or $r_{c,\succeq}(\beta) - 1 = \max_{\gamma \in Dom(\beta)} r_{c,\succeq}(\gamma)$. This cannot be the case because $\alpha$ is in $Dom(\beta)$, which means that $r_{c,\succeq}(\alpha) = \max_{\gamma \in Dom(\beta)} r_{c,\succeq}(\gamma)$. This is a contradiction. □

**Lemma 12.** *Given a partial preorder $\succeq$ if $\alpha \bowtie \beta$ and there exists a $\gamma$ such that $r_{c,\succeq}(\alpha) > r_{c,\succeq}(\gamma) > r_{c,\succeq}(\beta)$ then $\alpha \bowtie \gamma$, $\beta \bowtie \gamma$, or both hold.*

*Proof.* If neither $\alpha \bowtie \gamma$ or $\beta \bowtie \gamma$ hold then a relation exists between $\alpha$ and $\gamma$ and between $\beta$ and $\gamma$. Since we know that $r_{c,\succeq}(\alpha) > r_{c,\succeq}(\gamma) > r_{c,\succeq}(\beta)$ these relations must be $\alpha \succ \gamma$ and $\gamma \succ \beta$. This means that, by transitivity, $\alpha \succ \beta$ which cannot be the case since $\alpha \bowtie \beta$, a contradiction. □

Using the canonical ranking function and Lemmas 10, 11, and 12 we are able to prove that for any given preorder there is a preference theory which induces it.

**Proposition 1.** *Given a preorder $\succeq$ over a combinatorial domain $\mathcal{C}(\mathcal{V},\mathcal{D})$ it is possible to construct a preference theory $P$ such that for any pair of alternatives $\alpha,\beta \in \mathcal{C}(\mathcal{V},\mathcal{D})$ if $\alpha \succeq \beta$ then $\alpha \succeq_P \beta$.*

*Proof.* By Lemma 10 we know that if we can build a set of satisfaction vectors $S$, with one entry for each alternative in $\mathcal{C}(\mathcal{V},\mathcal{D})$, such that $s(\alpha) \geq_{\text{Pareto}} s(\beta)$ if $\alpha \succeq \beta$ then there exists a preference theory which induces $\succeq$ as its associated preorder. For our purposes we will consider the alternatives in $\mathcal{C}(\mathcal{V},\mathcal{D})$ to be split into clusters by the equivalence classes defined by the $\approx$ derived from $\succeq$. When we refer to alternatives below we really mean an equivalence class of alternatives.

We begin our construction by setting the first element of each satisfaction vector to the value of the canonical ranking function, $r_{c,\succeq}$, for that alternative. This means that, just considering the first element, all dominance relations in $\succeq$ are satisfied. This does not hold for incomparable alternatives relations, but as long as inserting incomparabilities does not remove these dominance satisfying properties we only need to worry about pairs of alternative $(\alpha,\beta)$ such that $\alpha \bowtie \beta$.

For each incomparable pair $(\alpha,\beta)$ induced by $\succeq$ we follow the process detailed below:

- Append $1,2$ to the satisfaction vector for $\alpha$.
- Append $2,1$ to the satisfaction vector for $\beta$.
- Append $1,1$ to the satisfaction vector for any alternative $\gamma$ such that $r_{c,\succeq}(\gamma) < r_{c,\succeq}(\alpha)$
- Append $2,2$ to the satisfaction vector for any alternative $\gamma$ such that $r_{c,\succeq}(\gamma) > r_{c,\succeq}(\beta)$
- Append $1,1$ to the satisfaction vector for any alternative $\gamma$ such that $r_c c, \succeq(\alpha) < r_{c,\succeq}(\gamma) < r_{c,\succeq}(\beta)$ and $\alpha \not\bowtie \gamma$.

- Append $2, 2$ to the satisfaction vector for any alternative $\gamma$ such that $r_c c, \succeq(\alpha) < r_{c,\succeq}(\gamma) < r_{c,\succeq}(\beta)$ and $\alpha \bowtie \gamma$.

For this construction we assume, without loss of generality, that $r_{c,\succeq}(\alpha) < r_{c,\succeq}(\beta)$.

Using this construction any two incomparable alternatives now have Pareto incomparable satisfaction vectors, since they each have at least one element strictly better than the other. Importantly, if there is a more preferred alternative than the pair each added element is at least as preferred as the ones added to the pair, thus preserving the dominance relation. Similar can be said for alternatives which are less preferred than the pair.

For alternatives which have a canonical rank in between those of the pair we know, by Lemma 12, they must be incomparable with at least one of the alternatives in the pair. The elements added to the satisfaction vectors of "in between" alternatives are such that dominance, if it exists, will still be satisfied after they are added.

Thus the constructed set of satisfaction vectors is such that $s(\alpha) \geq_{\text{Pareto}} s(\beta)$ if $\alpha \succeq \beta$ holds. This means that for any arbitrary preorder $\succeq$ there is a preference theory $P$ such that for any pair of alternatives $\alpha, \beta \in \mathcal{C}(\mathcal{V}, \mathcal{D})$ if $\alpha \succeq \beta$ then $\alpha \succeq_P \beta$. $\square$

The above proofs show the validity of all the edges shown in Figure 3.1. It is important that we also prove that for each missing edge/path between a pair of languages there is no subsumption relation and if there is a one-way edge that the languages are not equivalent. To our knowledge such analysis has not been done in the literature and we prove lack of subsumption between various languages below.

**Lemma 13.** *CICP LP-Tree $\not\subseteq$ CIUP LP-Trees*

*Proof.* Consider a CICP LP-tree $T$ which has a single conditional preference on attribute $A$, which is conditioned by attribute $B$. Suppose we build a CIUP LP-tree $T'$ which is identical to $T$, minus the conditional preference. This means we set a preference order for values of $A$ which is independent of the value of $B$. We then construct two alternative $\alpha$ and $\beta$ by setting $\alpha[x] = \beta[x]$ for every attribute $x$ in the domain (except for $A$). Suppose $T'$ correctly decides $\succeq_T$ for $\alpha$ and $\beta$. If we then change the value of $\alpha[B]$ and $\beta[B]$ such that the preference over $A$ changes in $T$ then $T'$ will induce the incorrect relation between the new pair of alternatives. This is because $T'$ does not have conditional preferences and so must always choose one of the multiple possible preference orders for $A$ given by $T$ and thus cannot induce a preorder identical to $\succeq_T$ in all cases. This means there exists CICP LP-tree models which cannot be replicated by a CIUP LP-tree model. $\square$

**Lemma 14.** *CICP LP-Tree $\nsubseteq$ UICP LP-Tree*

*Proof.* Consider a CICP LP-tree $T$ which has conditional importance such that after considering attribute $A$, if $\alpha[A] = 0$ then attribute $B$ is considered. If $\alpha[A] \neq 0$ then attribute $C$ is considered. Suppose there existed a UICP LP-tree $T'$ such that $\succeq_{T'}$ was identical to $\succeq_T$. Consider the set of alternatives who differ only on attributes $A$, $B$, and $C$ such that all possible combinations of values for $A$, $B$, and $C$ are included and each alternative in the set is unique. Among the set there must exist a pair $\alpha$ and $\beta$ such that $\alpha \succeq_T \beta$ which is decided on $B$ after considering $A$ and $\beta[C]$ is preferred to $\alpha[C]$ in $T'$. This means that if $\alpha \succeq_{T'} \beta$ then after $A$ is considered $B$ must be considered in $T'$. There must also be another pair $\alpha'$ and $\beta'$ such that $\beta' \succeq_T \alpha'$ which is decided on $C$ and $\alpha'[B]$ is preferred to $\beta'[B]$ in $T'$. This means that if $\beta' \succeq_{T'} \alpha'$ then after $A$ is considered $C$ must be considered in $T'$. The cases of $(\alpha, \beta)$ and $(\alpha', \beta')$ cannot both be true since $T'$ cannot change the order in which attributes are considered. Therefore, there must exist a CICP LP-tree model whose order cannot be induced by a UICP LP-tree model. $\square$

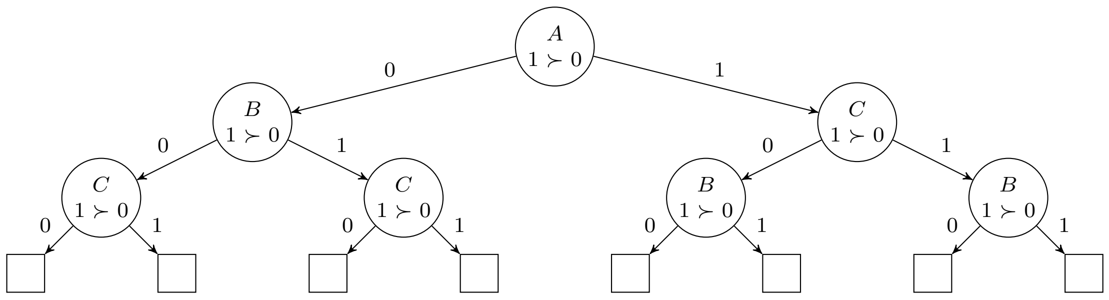
**Lemma 15.** *UICP LP-Tree $\nsubseteq$ LPM*

*Proof.* Consider a UICP LP-tree $T$ which has a single conditional preference on an attribute $A$, which is conditioned by attribute $B$. Suppose we build an LPM $\pi$ by setting the importance order of $\pi$ is identical to a traversal of $T$ from its root to a leaf, which is always the same no matter which branches we choose because $T$ has no conditional importance. Therefore, $\pi$ contains the same importance order of attributes as $T$. Since $\pi$ is an LPM we must set the preference orders for each attribute in an unconditional manner. Whatever preference order we choose for values of $A$ in $\pi$, that order will be independent of the value of $B$ when considering alternatives. We construct two alternative $\alpha$ and $\beta$ by setting $\alpha[x] = \beta[x]$ for every attribute $x$ in the domain (except for $A$). Suppose $\pi$ correctly decides $\succeq_T$ for $\alpha$ and $\beta$. If we then change the value of $\alpha[B]$ and $\beta[B]$ such that the preference over $A$ changes according to $T$ then $\pi$ will induce the incorrect relation between the new pair of alternatives. Since $\pi$ does not have conditional preferences its preference order for $A$ is static meaning it cannot induce the same order as $\succeq_T$. This means there exists models in the UICP LP-tree language which induce preorders that cannot be induced by a model in the LPM language. □

**Lemma 16.** *CIUP LP-Trees $\not\subseteq$ LPM*

*Proof.* Consider a CIUP LP-tree $T$ which has conditional importance such that after considering attribute $A$, if $\alpha[A] = 0$ then attribute $B$ is considered. If $\alpha[A] \neq 0$ then attribute $C$ is considered. Suppose there existed an LPM $\pi$ such that $\succeq_\pi$ was identical to $\succeq_T$. Consider the set of alternatives who differ only on attributes $A$, $B$, and $C$ such that all possible combinations of values for $A$, $B$, and $C$ are included and each alternative in the set is unique. Among the set there must exist a pair $\alpha$ and $\beta$ such that $\alpha \succeq_T \beta$ which is decided on $B$ after considering $A$ and $\beta[C]$ is preferred to $\alpha[C]$ in $\pi$. This means that if $\alpha \succeq_\pi \beta$ then after $A$ is considered $B$ must be considered in $\pi$. There must also be another pair $\alpha'$ and $\beta'$ such that $\beta' \succeq_T \alpha'$ which is decided on $C$ and $\alpha'[B]$ is preferred to $\beta'[B]$ in $\pi$. This means that if $\beta' \succeq_\pi \alpha'$ then after $A$ is considered $C$ must be considered in $\pi$. The cases of $(\alpha, \beta)$ and $(\alpha', \beta')$ cannot both be true since $\pi$ cannot contain conditional importance, therefore there must

Figure 3.2: A CIUP LP-tree whose preorder cannot be induced by a model in Weighted Average language.



exist a model in the CIUP LP-tree language which induces a preorder that cannot be induced by a model of the LPM language. ☐

**Lemma 17.** *Weighted Average Model $\not\subseteq$ LPM*

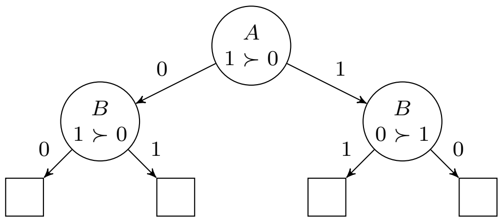
*Proof.* A model in the LPM language is incapable of producing $\approx$ relations and thus by Lemma 1 the LPM language cannot subsume Weighted Average Models language. ☐

**Lemma 18.** *Utility Function $\not\subseteq$ Weighted Average Model*

*Proof.* Given that for any total preorder there exists a ranking function which induces the total preorder, by Lemma 6, and the Utility Function language subsumes the Ranking Function language, by Lemma 8, if a weighted average model cannot replicate any arbitrary total preorder, the Weighted Average Model language cannot subsume the Utility Function language. Consider the total order $\succeq$ which induces the order $11 \succ 00 \succ 01 \succ 10$. Suppose there existed a weighted average preference model which induces $\succeq$ using the weights $(w_1, w_2)$. We know the relative value of the utility function must be as follows: $u(11) > u(00) > u(01) > u(10)$ or $(w_1 + w_2) > 0 > w_1 > w_2$. It follows that $w_1, w_2 < 0$ and $w_1 + w_2 > 0$, a contradiction. ☐

**Lemma 19.** *CIUP LP-Trees $\not\subseteq$ Weighted Average Model*

Figure 3.3: A UICP LP-tree whose induced preorder cannot be induced by a model in the Weighted Average language.



*Proof.* Consider the CIUP LP-tree given in Figure 3.2, where we express alternatives as a tuple following the order $ABC$. It induces the following order $111 \succ 101 \succ 110 \succ 100 \succ 011 \succ 010 \succ 001 \succ 000$. When building a weighted average model to induce this order we know that $1 \succ 0$ since in all cases we have $1 \succ 0$ according to the LP-tree. This means the weighted average must satisfy the following inequality: $w_A + w_B + w_C > w_A + w_C > w_A + w_B > w_A > w_B + w_C > w_B > w_C > 0$. Observe that $w_B > w_C$ is directly shown in the inequality. If we look at another comparison $w_A + w_C > w_A + w_B$ we can simplify this inequality down to $w_C > w_B$. This means there are no values for $w_B$ and $w_C$ which could induce the preference order given by the LP-tree and so CIUP LP-trees are not subsumed by Weighted Averages. □

**Lemma 20.** *UICP LP-Tree* $\not\subseteq$ *Weighted Average*

*Proof.* Consider the UICP LP-tree given in Figure 3.3, where we express alternatives as a tuple following the order $AB$. It induces the following order $10 \succ 11 \succ 01 \succ 00$. Consider building a weighted average to induce the same order. We know that $1 \succ 0$ for $A$ in all cases, but we can choose either $1 \succ 0$ or $0 \succ 1$ for the order produced by the weighted average. If $1 \succ 0$ is chosen for $B$ the utility of 11 is $w_A + w_B$, the utility of 10 is $w_A$, the utility of 01 is $w_B$, and the utility of 00 is 0. In this case we have that $w_A > w_A + w_B > w_B > 0$. It is impossible for $w_B > 0$ and $w_A > w_A + w_B$ so the weighted average cannot have $1 \succ 0$ for $B$. If $0 \succ 1$ is chosen for $B$ the utility of 11 is $w_A$, the utility of 10 is $w_A + w_B$, the utility of 01 is 0, and the utility of 00 is $w_B$. In this case we have that $w_A + w_B > w_A > 0 > w_B$. It is impossible for $0 > w_B$ and $w_A + w_B > w_A$. Therefore, there is no preference order over $B$ that a weighted average model can have such that that model would induce the same preorder as the given UICP LP-tree. This means that there are models in the UICP LP-tree language whose induced orders cannot be induced by a model of the Weighted Average language. □

**Lemma 21.** *CICP LP-Tree* $\not\subseteq$ *Acyclic CP-nets*

*Proof.* CICP LP-Tree are incapable of producing $\bowtie$ relations and thus by Lemma 1 the CICP LP-Tree language does not subsume the Acyclic CP-net language. □

**Lemma 22.** *LPM* $\not\subseteq$ *Acyclic CP-nets*

*Proof.* By definition any two adjacent elements in a preference order defined by an acyclic CP-net must differ by a single element. Consider an LPM over a domain with three binary attributes $A, B, C$ where $1 \succ 0$ for each attribute and the importance order $A \succ B \succ C$. This induces the order $111 \succ 110 \succ 101 \succ 100 \succ 011 \succ 010 \succ 001 \succ 000$. As we can see 100 and 011 are adjacent in the order induced by the LPM, which cannot be induced by an Acyclic CP-net, thus the LPM language is not subsumed by the Acyclic CP-net language. □

**Lemma 23.** *CIUP LP-Tree* $\not\subseteq$ *CLPM*

*Proof.* Due to how a CLPMs preference relation is defined there can be no two attributes with the same importance in any CLPM which attempts to reproduce a CIUP LP-tree given that in CIUP LP-trees no two alternatives are incomparable. This means we are left with those CLPMs which induce identical preorders to UICP LP-trees, thus because the UICP LP-tree langauges does not subsume the CIUP LP-tree language we know that the CLPM language does not subsume the CIUP LP-tree language. □

**Lemma 24.** *Weighted Average Model $\not\subseteq$ CLPM*

*Proof.* CLPMs are incapable of producing $\approx$ relations and thus by Lemma 1 the CLPM language does not subsume the Weighted Average language. $\square$

**Lemma 25.** *CLPM $\not\subseteq$ Weighted Average Model*

*Proof.* Weighted Average Models are incapable of producing $\bowtie$ relations and thus by Lemma 1 the Weighted Average language does not subsume the CLPM language. $\square$

**Lemma 26.** *Weighted Average Model $\not\subseteq$ Acyclic CP-nets*

*Proof.* Weighted Average Models are incapable of producing $\bowtie$ relations and thus by Lemma 1 the Weighted Average language does not subsume the Acyclic CP-net language. $\square$

Figure 3.4: A CP-Net

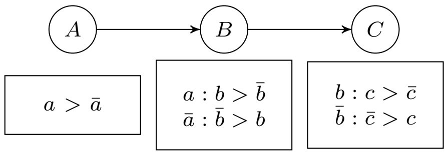
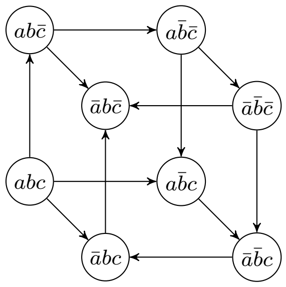


Figure 3.5: The worsening flip relation $\rightarrow_C$ for the CP-net in Figure 3.4



**Lemma 27.** *Acyclic CP-nets $\not\subseteq$ CLPM*

*Proof.* Consider the acyclic CP-net $C$ in Figure 3.4 which produces the preorder given by the preference graph in Figure 3.5. Consider a CLPM $\pi$ constructed which has the same conditional preference tables. If $\pi$ does not have the same CPTs as $C$ then some $\succ_C$ relations would not be induced by $\pi$. Since $\pi$ has the same CPTs as $C$ it must have an importance ranking over the attributes which induces a total order. If a CLPM has a total order induced by its importance ranking then it is a UICP LP-tree. Since the language of UICP LP-trees does not subsume the Acyclic CP-net language then the CLPM language does not subsme the Acyclic CP-net language. □

These lack of subsumption proofs are sufficient, when combined with the given proofs of subsumption, to produce the graph given in Figure 3.1. Any edges, and lack of edges, not proven here can be derived from the proven relations using transitivity.

## 3.4 Comparing Properties and Semantics

The proofs used in our case study of subsumption show some interesting uses for analyzing languages. For one, subsumption creates a preorder over the set of preference representation languages. This ordering allows us to compare how expressive those languages are. For example we know that, of the languages considered, PTs are the most expressive overall, while LPMs are among the least expressive.

Our subsumptive analysis also shows us, primarily through proofs that two languages do not share a subsumptive relationship, that there are significant trade-offs between features of a language and what preorders that language can express. For example, we see that by limiting the type of utility function to a weighted average the number of expressible orders is reduced. Similarly, we find that for LP-trees conditional importance and conditional preference cause an increase in expressive power over LPMs, and that conditional importance and conditional preference represent two different ways to extend expressive power, i.e. adding one is not the same as adding the other.

Finally, subsumptive analysis allows us to determine which language features are redundant. If we take a look at our proof that there is a least one PT which induces any given arbitrary preorder, we see that the program's formula do not need to be ranked. While this feature may be useful in producing more human-readable or compact PTs they do not add additional expressive power, since preorders are the least restrictive type of relation we deal with.

In general, we find that there is a trade-off between the limit on the size of a model and what can be expressed by models of that size. Looking back to the subsumption proofs above we can see that many languages, specifically those capable of expressing any possible total preorder, or partial preorder, involve non-compact representations. Specifically, the proof showing ranking preference formulas are capable of expressing any arbitrary ranking function requires that we have one formula for each alternative. Since combinatorial domains are exponential in size, wrt to the number of attributes, the resultant RPF would also be exponentially large (although, in some settings, it may be possible to produce more compact models through logical transformations). As we move to less expressive preference languages, such as weighted averages and LPMs we find that models become relatively compact, that is polynomially sized wrt the number of attributes in the domain. This implies that the more expressive a language the less compact its models may be.

## 3.5 Limitations

While a useful tool for analyzing preference representation languages subsumption also has its limitations. First and foremost subsumption is a qualitative measure. Just as we lose some expressive power when converting from a utility function to a ranking function, subsumption disregards potential nuance between models in different languages as long as they induce the same preference relations. On this note it is important to point out that subsumption acts as an "all or nothing" relation.

This means that even if two languages are capable of producing equivalent models except for a single case (on each side), then the two languages will be rendered incomparable. We see this with CLPMs and CP-nets. There is a significant amount of overlap between CLPMs and CP-nets in terms of equivalent models [Huelsman and Truszczyński, 2020], which we will discuss in more detail later. In fact, one can approximate dominance in an acyclic CP-net using a CLPM with relatively high accuracy. The subsumption relation does not deal with such nuance and so the languages are incomparable wrt subsumption.

Another problem with subsumption being a qualitative approach is showcased by the relation between utility functions and ranking functions. According to subsumption the languages are similar, however there is a loss of information when converting a utility function to a ranking function. This is due to utility functions being quantitative while ranking functions are qualitative.

A utility function produces a utility value for each alternative. The values not only allow us to say which alternative is preferred but also how much more preferred it is. This latter piece of information is absent from qualitative approaches, such as ranking functions. For example if we have three objects $A$,$B$, and $C$ and a utility function gives them value 0.5, 0.4, and 0.1, respectively, then we know that while $A$ is preferred to $B$ and $B$ is preferred to $C$, the difference between $A$ and $B$ is much smaller than that of $A$ and $C$. In practical terms a person might not mind being given $B$ instead of $A$, but $C$ would be insulting. The ranking function defined by: $r(A) = 1, r(B) = 2$, and $r(C) = 3$ determines the same order, but provides no information about the degree to which one alternative is preferred to another. This limitation indicates that subsumption should not be the sole consideration when comparing two languages.

Finally, a general limitation of subsumption comes from how we choose to define a preference representation language worth considering. For example, LP-trees can be considered as a single language or we could break it down into several component languages based on conditional importance and conditional preferences, as we do in Figure 3.1. We might also wish to separate RPFs by the number of ranks they contain or absorb weighted average models into utility functions. In the most extreme case we could make these components only contain a very small number of models, which would make analysis time consuming and tedious. On the other hand, too few components and the analysis becomes too broad to be useful. While we chose to break the languages up as we did, other categorizations might be useful for other applications.

# Chapter 4 Case Study: Approximating CP-nets with CLPMs

The interaction between the preference representation languages of acyclic CP-nets and conditional lexicographic preference models (CLPMs) showcases one of the limitations of subsumptive analysis. Namely, the inability of subsumption to identify languages whose models might make for good approximators of another language's models. In the case of acyclic CP-nets and CLPMs, CLPMs perform well as an approximation of the preorders induced by acyclic CP-nets, but the languages are incomparable by subsumption.

CP-nets provide an intuitive, principled approach for their construction, encode conditional preference information, and allow for polynomial time computation of optimal and pessimal alternatives [Boutilier et al., 2004]. CP-nets have one primary disadvantage: computing dominance can be computationally difficult. In the case where an acyclic CP-net's dependency graph does not form a tree, or poly-tree, dominance is NP-hard [Boutilier et al., 2004]. With general CP-nets the problem becomes PSPACE-complete [Goldsmith et al., 2008].

In order to compute dominance more efficiently for acyclic CP-nets we propose converting CP-nets into CLPMs and then computing dominance using the CLPM. This conversion allows us to approximate the dominance relation of an acyclic CP-net in polynomial time. While the approximating CLPM of a CP-net does not always induce the same preorder as the CP-net it preserves all strict preference relations in the original CP-net's induced preorder.

## 4.1 Constructing an Approximating CLPM

A CP-net approximating CLPM consists of a pair $(C, r)$ where $C$ is the CP-net $(\mathcal{V}, E, T)$ being approximated and $r$ is a ranking on $\mathcal{V}$ consistent with attribute dependencies in $C$, that is, if edge $(u, v)$ appears in the dependency graph of $C$ then $r(u) < r(v)$. Any valid CLPM consisting of such a pair is an approximating CLPM of $C$. In general, there may be more than one ranking that is consistent with $C$, so a single CP-net $C$ may have several approximating CLPMs. We denote the family of all CLPMs which approximate a given CP-net $C$ as $\mathit{CLPM}(C)$.

Of course different members of $\mathit{CLPM}(C)$ provide different approximations and therefore different levels of accuracy. Since we want our approximation to be as close as possible to the original CP-net we need to know which approximating CLPMs are most accurate. To this end we prove some useful properties of approximating CLPMs. First, we show that each CLPM in $\mathit{CLPM}(C)$ induces the single flip preferences of $C$, see Lemma 28.

**Lemma 28.** *Given a CP-net $C = (\mathcal{V}, E, T)$ and a CLPM, $\pi \in \mathit{CLPM}(C)$, for every pair of alternatives $\alpha$ and $\beta$ such that $\alpha$ differs from $\beta$ on a single attribute, if $\alpha \succ_C \beta$ then $\alpha \succ_\pi \beta$.*

*Proof.* Since $\alpha \succ_C \beta$ and they differ by a single attribute we know that $\alpha \succ_T \beta$ also holds. We designate the differing attribute as $v$. Since $\alpha[\mathcal{V} \setminus \{v\}] = \beta[\mathcal{V} \setminus \{v\}]$ the rank of $v$ does not matter, as it is the only place where a dominance decision could possibly be made. Since $\alpha[v] \succ_{v,\alpha} \beta[v]$ we have that $\alpha \succ_\pi \beta$. □

Lemma 28 allows us to show that the dominance relation $\succ_\pi$ for a CLPM $\pi \in \mathit{CLPM}(C)$ is an overapproximation of the dominance relations $\succ_C$ for a CP-net $C$.

**Theorem 1.** *Given a CP-net $C = (\mathcal{V}, E, T)$ and a CLPM $\pi \in \mathit{CLPM}(C)$, for every two alternatives $\alpha$ and $\beta$, if $\alpha \succ_C \beta$ then $\alpha \succ_\pi \beta$.*

*Proof.* Consider an arbitrary worsening flipping sequence of alternatives, according to $C$: $o_1, o_2, o_3, \cdots, o_k$. This means that $o_i \rightarrow_C o_{i+1}$, and by transitivity $o_1 \succ_C o_k$. Since $\pi$ is an approximating CLPM we know by Lemma 28 that $o_i \succ_\pi o_{i+1}$. Furthermore, given that $\succ_\pi$ is transitive we know that $o_1 \succ_\pi o_k$. □

In general, the overapproximation of an approximating CLPM is a proper one, that is, some pairs in the dominance relation of $\pi \in \mathit{CLPM}(C)$ are not in the dominance relation of the CP-net $C$. For example, it is possible for a pair of alternatives $(\alpha, \beta)$ that a CLPM $\pi \in \mathit{CLPM}(C)$ induces $\alpha \succ_\pi \beta$ while the CP-net $C$ induces $\alpha \bowtie_C \beta$. Furthermore, Theorem 1 means that if a CLPM $\pi \in \mathit{CLPM}(C)$ induces $\alpha \bowtie_\pi \beta$ then we know that $\alpha \bowtie_C \beta$, see Corollary 1.

**Corollary 1.** *Given a CP-net $C = (\mathcal{V}, E, T)$ and a CLPM $\pi \in \mathit{CLPM}(C)$, for every two alternatives $\alpha$ and $\beta$, if $\alpha \bowtie_\pi \beta$ then $\alpha \bowtie_C \beta$.*

*Proof.* Given two alternatives $\alpha$ and $\beta$, by Theorem 1, if $\alpha \not\succ_\pi \beta$ and $\beta \not\succ_\pi \alpha$ then $\alpha \not\succ_C \beta$ and $\beta \not\succ_C \alpha$. Thus, $\alpha \bowtie_C \beta$. $\square$

For a given acyclic CP-net $C$ this corollary show that if any CLPM in $\mathit{CLPM}(C)$ cannot induce a dominance relation between a pair of alternatives $\alpha$ and $\beta$ then $\alpha \bowtie_C \beta$. Given that an approximating CLPM overapproximates the CP-net's dominance relation, if some two alternatives are incomparable wrt any CLPM in $\mathit{CLPM}(C)$, then those two alternatives are incomparable wrt $C$. Thus, the intersection of the dominance relations of all CLPMs in $\mathit{CLPM}(C)$ offers the best approximation of $C$ that can be derived from the dominance relations of CLPMs in $\mathit{CLPM}(C)$. This brings us to the question of how to compute such an intersection.

## 4.2 Computing the Aggregated CLPM Approximation of a CP-net

Before we can compute the preference relation defined by the intersection of all models in $\mathit{CLPM}(C)$, which we will refer to as the aggregated $\mathit{CLPM}(C)$ approximation, or simply the aggregated approximation, we must define the intersection formally, see Definition 11.

**Definition 11** (Aggregated $CLPM(C)$ Approximation)**.** *Given a CP-net $C$ we define the aggregated approximation $\succ_{CLPM(C)}$ of $\succ_C$ by setting $\alpha \succ_{CLPM(C)} \beta$ if for all $\pi \in CLPM(C)$ we have that $\alpha \succ_\pi \beta$.*

Combining Definition 11 with Theorem 1 it is easy to show that for a given CP-net if $\alpha \succ_C \beta$ then $\alpha \succ_{CLPM(C)} \beta$. It also follows that if $\alpha \bowtie_{CLPM(C)} \beta$ then $\alpha \bowtie_C \beta$. Of course if two models induce inverse relations, $\succ$ and $\prec$, over a pair of alternatives then they will be incomparable wrt the aggregated approximation. Below in Theorem 2 we show that when two CLPMs in $CLPM(C)$ disagree on which alternative in a pair is preferred there exists another CLPM in $CLPM(C)$ such that the alternatives are incomparable.

**Theorem 2.** *Given a CP-net $C = (\mathcal{V}, E, T)$, two approximating CLPMs $\pi = (C, r)$ and $\pi' = (C, r')$, and two alternatives $\alpha$ and $\beta$, if $\alpha \succ_\pi \beta$ and $\beta \succ_{\pi'} \alpha$ then there exists an approximating CLPM $\pi'' = (C, r'')$ such that $\alpha \bowtie_{\pi''} \beta$.*

*Proof.* For determining the relation between $\alpha$ and $\beta$ consider the deciding ranks to be $i$ and $j$ for $\pi$ and $\pi'$, respectively. This means there exists attributes $v, w \in \mathcal{V}$ such that $r(v) = i$, $r'(w) = j$, $\alpha[v] \succ \beta[v]$, and $\beta[w] \succ \alpha[w]$. Since $r(v) < r(w)$ and $r'(w) < r'(v)$ which are both consistent ranking with $C$ there cannot be a path between $v$ and $w$ in the dependency graph or both attributes would be ordered the same. This means there must be a consistent ranking $r''$ such that $r''(v) = r''(w)$. We know that for any attributes $b$ such that $r(b) < r(v)$ or $r'(b) < r'(w)$, $\alpha[b] = \beta[b]$ so it follows that for $\pi'' = (C, r'')$, $\alpha \bowtie_{\pi''} \beta$. □

In other words, we do not need to consider every model in $CLPM(C)$ to compute the aggregated approximation. This means it may be computationally efficient to compute the aggregated approximation even when $CLPM(C)$ is large. The size of $CLPM(C)$ depends on the structure of the CP-net, but if we consider a separable CP-net, that is, its dependency graph has no edges, then any ranking applied to the CP-net would be a valid approximating CLPM. This means that in some cases $|CLPM(C)|$ has a lower bound of $n!$, where $n$ is the number of attributes. Thus, there are cases where brute force calculation of the aggregated approximation are intractable. A potential solution is to find a minimal subset of CLPMs which can be used to compute the aggregated approximation.

Models in $CLPM(C)$ are differentiated by their ranking and so if we consider how these rankings are related it is possible to reduce the number of CLPMs needed to compute the aggregated approximation of $CLPM(C)$. One way to eliminate models from consideration is to consider rankings which are *strict extensions* of each other, see Definitions 12.

**Definition 12** (Strict Ranking Extension)**.** *A ranking $r'$ of elements in a set $U$ is a strict extension of a ranking $r$ of $U$ if for all pairs of elements $a, b \in U$ if $r(a) < r(b)$ then $r'(a) < r'(b)$ and for some pair of elements $a', b' \in U$ where $r(a) = r(b)$, $r'(a) < r'(b)$.*

We show that when a ranking is extended dominance relations are preserved wrt to its approximating CLPM, see Theorem 3, and when contracted, the inverse of a strict rank extension, a ranking preserves incomparability wrt to its approximating CLPM, see Corollary 2. In other words, a ranking produces a more accurate approximating CLPM than its extensions.

**Theorem 3.** *Let $\pi = (C, r)$ be a CLPM in $CLPM(C)$ and let $r'$ be a strict extension of $r$. Then $\pi' = (C, r')$ is a CLPM in $CLPM(C)$ and, if $\alpha \succ_{\pi} \beta$ then $\alpha \succ_{\pi'} \beta$ holds, too.*

*Proof.* The proof is as follows:

- If $\pi \in CLPM(C)$ then $\pi' \in CLPM(C)$

  Since $r'$ is an extension of $r$ we know that if $r(v_i) < r(v_j)$ then $r'(v_i) < r'(v_j)$.

By the definition of a consistent ranking, wrt a CP-net, if $r$ is consistent with $C$ than $r'$ is also consistent with $C$ meaning $\pi' \in CLPM(C)$.

- If $\alpha \succ_{\pi} \beta$ then $\alpha \succ_{\pi'} \beta$

  If $\alpha \succ_{\pi} \beta$ then there is an attribute $a$ such that $\alpha[a] \succ \beta[a]$, for every attribute $b$ such that $r(b) < r(a)$, $\alpha[b] = \beta[b]$, and for every attribute $b$ such that $r(b) = r(a)$, $\alpha[b] \geq \beta[b]$. If $b$ is any attribute such that $r'(b) < r'(a)$ then $r(b) \leq r(a)$ and so, $\alpha[b] \geq \beta[b]$. If $r'(b) \geq r'(a)$ then $r(b) = r(a)$ or $r(b) > r(a)$. In each case $\alpha[a] \geq \beta[a]$. Thus, $\alpha \succ_{\pi} \beta$ follows.

□

**Corollary 2.** *Let $\pi = (C, r)$ be a CLPM in $CLPM(C)$ and let $r'$ be a strict extension of $r$. Then $\pi' = (C, r')$ is a CLPM in $CLPM(C)$ and, if $\alpha \bowtie_{\pi'} \beta$ then $\alpha \bowtie_{\pi} \beta$.*

*Proof.* The proof is as follows:

- If $\pi \in CLPM(C)$ then $\pi' \in CLPM(C)$

  Since $r'$ is an extension of $r$ we know that if $r(v_i) < r(v_j)$ then $r'(v_i) < r'(v_j)$. By the definition of a consistent ranking, wrt a CP-net, if $r$ is consistent with $C$ than $r'$ is also consistent with $C$ meaning $\pi' \in CLPM(C)$.

- If $\alpha \bowtie_{\pi'} \beta$ then $\alpha \bowtie_{\pi} \beta$

  If $\alpha \bowtie_{\pi'} \beta$ then we know that there must exist two attributes $a$ and $b$ such that $r'(a) = r'(b)$ and for the preference order of $a$, $\alpha \succ_a \beta$ and for the preference order of $b$, $\beta \succ_b \alpha$ (or vice versa). Since $r'$ is a strict extension of $r$ we know, by the definition of strict extension, that $r(a) = r(b)$ and for any attribute $c$ such that $r(c) < r(a)$, $r'(c) < r(a)$ holds meaning that all attributes more important than $a$ and $b$ according to $r$ must have the same value in $\alpha$ and $\beta$ or else they would not be incomparable according to $\pi'$. Thus, $\alpha \bowtie_{\pi} \beta$.

□

Since a ranking provides a better approximating CLPM than its extensions it must hold that a minimal ranking, one which cannot be contracted further, must provide the most accurate approximating CLPM of a given CP-net, for rankings that are extensions of that minimal ranking. We formally define a minimal consistent ranking as follows.

**Definition 13** (Minimal Consistent Ranking). *Given a CP-net $C = (\mathcal{V}, E, T)$ and a consistent importance ranking $r$, $r$ is minimal if there exists no ranking $r'$ such that $r'$ is also a consistent importance ranking and $r$ is an extension of $r'$.*

We conclude that the computation of the aggregated approximation requires only considering minimal rankings. We define the set of minimal consistent importance ranking approximating CLPMs, $MCLPM(C)$, to be the set of all minimal importance rankings such that all possible consistent importance rankings are either extensions of a ranking that is a member of $MCLPM(C)$ or a member themselves. In some cases, this greatly reduces the number of importance rankings under consideration. For example, when dealing with separable CP-nets we only need a single ranking (the ranking where all attributes have the same rank) since there are no ranking restrictions wrt the dependency graph.

**Definition 14** (Minimal Consistent Importance Rankings). *Given a CP-net $C = (\mathcal{V}, E, T)$ the set of minimal consistent importance rankings $MCLPM(C)$ consist of all CLPMs $\pi = (C, r)$ such that $r$ is a minimal consistent ranking wrt $C$.*

While the case of approximating separable CP-nets using the aggregated approximation becomes considerably more compact under this approach computing the aggregated approximation is still intractable in the general case. Consider a CP-net's dependency graph such that $\mathcal{V} = \{A, B, C\}$ and $E = \{(A, B)\}$. In this case there are two minimal rankings $A \succ \{B, C\}$ and $\{A, C\} \succ B$. If we expand this CP-net by adding $n-1$ singleton attributes, like $C$, then there are $2^n$ minimal rankings because we can assign each singleton attribute either equal to $A$ or equal to $B$. Any ranking of attributes that partitions between equality with $A$ and equality with $B$ is a minimal consistent ranking. This means the number of rankings in $MCLPM(C)$ are still, in general, intractable. Thus, using $MCLPM(C)$ does not represent a method for achieving a polynomial time algorithm for determining the aggregated approximation of an acyclic CP-net.

Despite the inability of ranking extensions to yield a tractable method of computing the aggregated approximation there does exist a polynomial time algorithm for computing the dominance relation induced by the aggregated approximation between two alternatives. The algorithm we propose takes three inputs: The CP-net to approximate, $C$, and a pair of alternatives $\alpha$ and $\beta$. For simplicity we assume that $\alpha$ and $\beta$ are different alternatives. The algorithm operates by attempting to find a partial importance order which induces $\alpha \bowtie \beta$. If such an importance order cannot be constructed then the importance order is used to construct an approximating CLPM $\pi$ and we return the relation between $\alpha$ and $\beta$ which is induced by $\pi$. This idea, with some additional improvements, is the basis for Algorithm 1.

Algorithm 1 does not consider all possible consistent importance rankings, nor does it leverage the relationships between the extensions of various rankings. The constructed importance ranking focuses on the attributes where the two alternatives differ and so the specific ranking generated depends on $\alpha$ and $\beta$. In other words, the algorithm tailors the approximating CLPM to find incomparability between those specific alternatives. Thus, the algorithm must be run on each pair of alternatives that one is interested in, rather than producing a singular model which can be reused. Moreover, we show that this approximation takes polynomial time, see Proposition 2.

**Algorithm 1** The aggregated lexicographic approximation of a CP-net $C$.

```
procedure LEX-EVAL-COMPARE($\alpha, \beta, C = (\mathcal{V}, E, T)$)
    $c \leftarrow 0$
    $r \leftarrow \emptyset$
    while $\exists v \in \mathcal{V}$ such that IN-DEG($v$) $= 0$ and $\alpha[v] = \beta[v]$ do
        for all $v \in \mathcal{V}$ where IN-DEG($v$) $= 0$ and $\alpha[v] = \beta[v]$ do
            Remove $v$ from $(\mathcal{V}, E)$
            $r(v) \leftarrow c$
            $c \leftarrow c + 1$
        end for
    end while
    $d \leftarrow \emptyset$
    for all $v \in \mathcal{V}$ where IN-DEG($v$) $= 0$ do
        $d \leftarrow d \cup \{v\}$
    end for
    $s \leftarrow \emptyset$
    for all $v \in d$ do
        if ($\alpha[v] \succ_{v,\alpha} \beta[v]$ and $s = \prec_{CLPM(C)}$) or ($\beta[v] \succ_{v,\alpha} \alpha[v]$ and $s = \succ_{CLPM(C)}$) then
            Return $\|_C$
        else if $\alpha[v] \succ_{v,\alpha} \beta[v]$ and $s = \emptyset$ then
            $s \leftarrow \succ_{CLPM(C)}$
        else if $\beta[v] \succ_{v,\alpha} \alpha[v]$ and $s = \emptyset$ then
            $s \leftarrow \prec_{CLPM(C)}$
        end if
    end for
    Return $s$
end procedure
```

**Proposition 2.** *LEX-EVAL-COMPARE takes time* $\mathcal{O}(|\mathcal{V}|^2)$

*Proof.* A topological sort can be done in linear time, with respect to the size of the graph. Thus the first part of the algorithm takes time $\mathcal{O}(|\mathcal{V}| + |E|)$

The next part of the algorithm compares $\alpha$ and $\beta$ over the elements in $d$. In the worst case $|d| = |\mathcal{V}|$. If all members of $d$ agree then all members must be considered and tested. If we assume that determining a comparison based on a CPT takes constant time then this part of the algorithm takes $\mathcal{O}(|\mathcal{V}|)$.

Adding the complexity of the two parts of the algorithm together we get that the overall time complexity for LEX-EVAL-COMPARE is $\mathcal{O}(|\mathcal{V}| + |E|)$, , with respect to the size of the dependency graph. □

While the algorithm runs in polynomial time, wrt the size of the CP-net, this does not tell us that the algorithm actually computes the aggregated CLPM approximation relation of a CP-net. In order to prove that we must first prove some properties of the algorithm. First, we show that the algorithm produces a consistent, if partial, importance ranking wrt the original CP-net (Lemma 29).

**Lemma 29.** *Lines* 4–14 *in LEX-EVAL-COMPARE produces a partial ranking consistent with the input CP-net $C$.*

*Proof.* LEX-EVAL-COMPARE builds its ranking by removing sets of attributes from the dependency graph which have an in-degree of 0. This means that if an attribute $v$ depends on another attribute $w$ then in order for $v$ to be given a rank $w$ must first be removed and given a lower rank. Thus $r(w) < r(b)$. This holds for any pair of dependent attributes and so the ranking produced must be consistent with the CP-net. While LEX-EVAL-COMPARE does not produce a complete ranking the remaining alternatives are not used in the decision to be made and thus can be ordered in any consistent manner without affecting the decision, and can be disregarded. □

The next property we show is that the algorithm determines the alternatives to be incomparable if and only if there exists a ranking such that the related approximating CLPM renders the alternatives incomparable. In other words, the algorithm will not render the alternatives incomparable if the aggregated relation would not as well. This is important because, by Theorem 2, all CLPMs in $CLPM(C)$ must agree on the relation between a pair of alternatives if the alternatives are not incomparable wrt the aggregated approximation. The formal result is proven below.

**Lemma 30.** *Given a fixed pair of alternatives $(\alpha, \beta)$ LEX-EVAL-COMPARE builds a consistent ranking $r$ such that for $\pi = (C, r)$ $\alpha \bowtie_\pi \beta$ if and only if there exists a CLPM $\pi' \in CLPM(C)$ with consistent ranking $r'$ such that $\alpha \bowtie_{\pi'} \beta$.*

*Proof.* The proof is as follows:

- $\alpha \bowtie_{\pi} \beta$ if $\alpha \bowtie_{\pi'} \beta$

  By Lemma 29 we know that $r$ is a consistent ranking. This means that $\pi$ is a member of $CLPM(C)$ and so if we set $\pi' = \pi$ the statement is trivially true.

- $\alpha \bowtie_{\pi'} \beta$ if $\alpha \bowtie_{\pi} \beta$

  By the definition of incomparability in CLPMs we know that for the deciding rank $i$ of $r'$ there must exist two attributes $v, w \in \mathcal{V}$ such that $\alpha[v] \succ_{v,\alpha} \beta[v]$ and $\beta[w] \succ_{w,\alpha} \alpha[w]$. Since $v$ and $w$ have different values in $\alpha$ and $\beta$ we know that they cannot appear in a more important rank than $d$, in $r$. If $v$ or $w$ are ranked as less important than $d$ then they must be conditionally dependent on another unranked attribute or on an attribute in $d$. This means that it would be inconsistent for $v$ or $w$ to be in the deciding rank. If this were the case then $r'$ would not be consistent as some attribute $c$ on which $\alpha$ and $\beta$ differ would need to be more important than $v$ or $w$ meaning neither $v$ or $w$ would be in the decision rank of $r'$ for $\alpha$ and $\beta$, which is a contradiction.

$\square$

Using Lemmas 29 and 30 we show that LEX-EVAL-COMPARE induces the dominance relation defined by the aggregated approximation for a particular CP-net for any arbitrary pair of alternatives. Thus it is possible to compute the aggregated approximation in polynomial time.

**Theorem 4.** *Given a CP-net $C = (\mathcal{V}, E, T)$ and two alternatives $\alpha$ and $\beta$, $\alpha \neq \beta$, LEX-EVAL-COMPARE correctly decides dominance according to the aggregated $CLPM(C)$ approximation.*

*Proof.* Given that $\alpha \prec_{CLPM(C)} \beta$ is equivalent to $\beta \succ_{CLPM(C)} \alpha$ we disregard $\prec_{CLPM(C)}$ and focus on the case of $\alpha \succ_{CLPM(C)} \beta$. We denote the relation induced by the algorithm as $\succ_A$.

$\alpha \succ_A \beta$ if and only if $\alpha \succ_{CLPM(C)} \beta$.

- If $\alpha \succ_{CLPM(C)} \beta$ then for any consistent ranking $r$ and corresponding CLPM $\pi = (C, r)$, $\alpha \succ_\pi \beta$. Since there exists no consistent ranking such that $\alpha$ and $\beta$ are incomparable, we know, by Lemma 30, that the algorithm will produce no ranking such that $\alpha$ and $\beta$ are incomparable.

- If the algorithm induces $\alpha \succ_A \beta$ then it produces a consistent ranking $r$ and corresponding CLPM $\pi = (C, r)$, such that $\alpha \succ_\pi \beta$. By Lemma 30 we know that a consistent ranking $r'$ and corresponding CLPM $\pi' = (C, r')$, does not exist such that $\alpha \bowtie_{\pi'} \beta$. Additionally, we know that there must not exist a consistent ranking $r''$ and corresponding CLPM $\pi'' = (C, r'')$, such that $\beta \succ_{\pi''} \alpha$ or else, by Theorem 2, a consistent ranking would exist where $\alpha$ and $\beta$ are incomparable. This means that all consistent rankings must agree and therefore $\alpha \succ_{CLPM(C)} \beta$.

$\square$

These results indicate it is possible to approximate a CP-net's dominance relation in polynomial time while preserving some pairs as incomparable, however the results do not indicate how "good" this approximation is. We investigate the accuracy of our approximation below.

## 4.3 Approximation Analysis

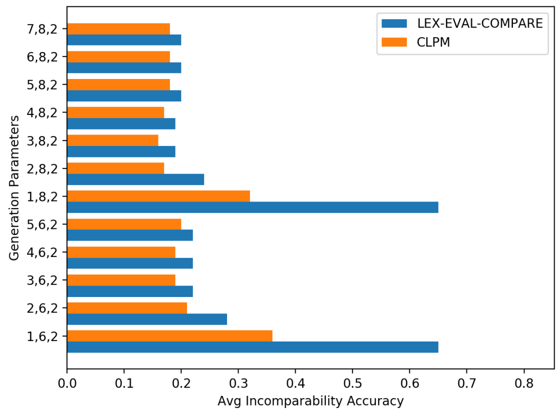
We have already shown that using CLPMs to approximate dominance in CP-nets induces a stronger dominance relation than that of the original CP-net. These results do not provide us with a way of efficiently calculating the accuracy of a given CP-net's approximation. In order to better understand how accurate our approximations are we performed an empirical analysis of the approximation.

Our experiments require the generation of acyclic CP-nets. For this purpose we selected a software tool from Allen et al. [2016]. This tool generates CP-nets uniformly at random and allows for controlling the upper bound on the indegree of attributes in the dependency graph. Simply put, we can build a set of random CP-nets where we limit the number of attributes which can condition any one attribute's preference order.

Despite the guarantee that our aggregated approximation is more accurate we also compute the accuracy of a single approximating CLPM. This allows us to determine if the increase in accuracy is, in general, beneficial or superficial. Of course, there may be many possible approximating CLPMs. We standardize the singular CLPM we consider to ensure consistency. For our canonical CLPM we chose to construct its ranking using a simple topological sorting algorithm. Specifically, we sorted the attributes by first removing all attributes in the dependency graph with no incoming edges and assigning them rank 1. We then assign the next group which has no incoming edges rank 2. This procedure continues until there are no more attributes in the graph. We chose to use this particular topological sort because it is easy to compute and attempts to maximize the size of each rank, thus maximizing the number of pairs of alternatives that are incomparable and therefore maximizing accuracy. We note that a similar sort could also be performed by removing vertices with no outgoing edges, although we do not study it here.

We measure the difference between a CP-net and its approximating CLPM by counting the number of times the approximating CLPM induces a $\succ$ or $\prec$ relation between two alternatives when the CP-net induces neither. We know from Theorem 1 and Corollary 1 that these are only types of mistakes made by our approximation. For simplicity, the data is reported as the proportion of correctly decided pairs. Note that we omit the case of separable CP-nets from our empirical analysis since we can produce a perfect approximation of any separable CP-net, which we will discuss

Figure 4.1: The average proportion of correctly decided incomparable pairs using LEX-EVAL-COMPARE



further later.

Figure 4.1 groups CP-nets based on the parameters used to generate them. CP-net groups are labeled with a triple $(i, j, k)$, where $i$ is the bound on the indegree, $j$ is the number of attributes and $k$ is the size of the attribute domains. The figure shows that the approximation performs worse as we increase the bound on the indegree, this indicates that more conditional information inhibits the ability of the approximation to accurately represent the original CP-net. Additionally, as the indegree bound increases there is a similar decrease in the difference between the accuracy of our canonical CLPM and LEX-EVAL-COMPARE, although the aggregated approximation is always more accurate. For the settings were CP-nets were generated with an indegree bound of 1, LEX-EVAL-COMPARE has nearly double the accuracy of the canonical CLPM. We conjecture that in these settings there is enough structure to the dependency graph that they generate large $MCLPM(C)$ sets. Since the canonical CLPM only handles a single ranking it is at a disadvantage compared to LEX-EVAL-COMPARE which uses the intersection of all possible approximating CLPMs.

## 4.4 Approximation of Special Cases

Since an approximating CLPM will always induce dominance relations that are induced by the original CP-net, we know that if a CP-net induces a total order then any approximating CLPM would also induce that total order. This means that all CP-nets which induce total orders can be perfectly approximated by a CLPM.

**Proposition 3.** *Given a CP-net $C = (\mathcal{V}, E, T)$ if $\succ_C$ induces a strict total order then $\succ_\pi$ is identical to $\succ_C$.*

*Proof.* For all alternative pairs $(\alpha, \beta)$ the total ordering $\succ_C$ either assigns $\alpha \succ_C \beta$ or $\beta \succ_C \alpha$ it must also be true, due to Theorem 1, that the ordering $\succ_\pi$ also induces these relations meaning all pairs of alternatives are compared identically. $\square$

CP-nets that induce total orders have a specific structure, namely their dependency graph must contain a Hamiltonian path, see Proposition 4. Since dominance occurs in CP-nets through the use of single value flips any total order must form a Gray code [Gilbert, 1958], i.e. each alternative in the linear order only differs on a single attribute when compared to its neighbors.

**Proposition 4.** *Given a CP-net $C = (\mathcal{V}, E, T)$ and an approximating CLPM $\pi = (C, r)$ if $\succ_C$ is a strict total ordering then $C$'s preference graph contains a Hamiltonian path.*

*Proof.* Suppose not. Suppose there existed an acyclic CP-net $C$ which induced the total ordering $\succ_C$ without a Hamiltonian path in its dependency graph. Since such a path does not exist we can build a ranking such that for some rank $i$, $|\mathcal{V}_i| > 1$. In this case we can construct two alternatives $\alpha$ and $\beta$ such that for all $v \in (\mathcal{V} \setminus \mathcal{V}_i)$ $\alpha[v] = \beta[v]$ and for some pair of attributes $w, x \in \mathcal{V}_i$ where $w \neq x$ $\alpha[w] \succ_{w,\alpha} \beta[w]$ and $\beta[x] \succ_{x,alpha} \alpha[x]$. By Definition 1 we know that neither $\alpha \succ_\pi \beta$, nor $\beta \succ_\pi \alpha$ can be derived and thus $\alpha \bowtie_\pi \beta$. This cannot be the case since $C$ is a total order and by Corollary 3 $\succ_\pi$ is the same as $\succ_C$. Thus we have a contradiction. $\square$

Figure 4.2: A CP-Net

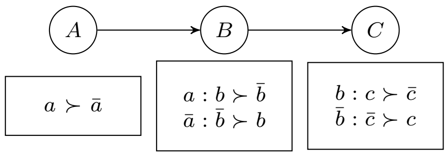
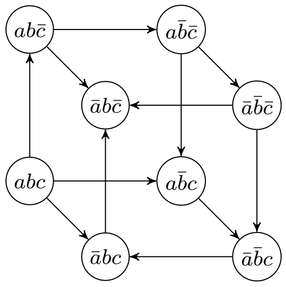


Figure 4.3: The worsening flip relation $\rightarrow_C$ for the CP-net in Figure 4.2



While a Hamiltonian path in the dependency graph is necessary for a CP-net to induce a total order it is not sufficient. Figure 4.2 shows a CP-net which contains a Hamiltonian path, but which does not induce a total order, the CP-net's induced order is shown in Figure 4.3. Along with total orders it is also possible for CLPMs to accurately approximate separable CP-nets, whose dependency graphs have no edges. In other words, when there is no conditional preference information, see Corollary 3.

**Corollary 3.** *If an acyclic CP-net $C$ has no attribute dependencies, then the approximating CLPM $\pi = (C, r)$ associated with a minimal consistent ranking $r$, where $r$ contains only a single rank, induces the same order as $C$.*

*Proof.* Consider a CP-net $C$ with no conditional dependencies, thus there are no edges in the dependency graph. Since $C$'s dependency graph has no edges the CLPM $\pi = (C, r)$ whose ranking function $r$ assigns the same value to all attributes is an approximating CLPM. In fact since the ranking cannot contract any further it is an approximating CLPM with a minimal ranking. We already know that or two alternatives $\alpha$ and $\beta$ if $\alpha \succ_C \beta$ then $\alpha \succ_\pi \beta$, by Theorem 1.

This means the only possible difference between $\succ_C$ and $\succ_\pi$ are when $\alpha \bowtie_C \beta$ occurs. Consider a pair of alternatives $(\alpha, \beta)$ such that $\alpha \bowtie_C \beta$. In order for $\alpha \bowtie_C \beta$ to occur there cannot be a worsening flipping sequence between $\alpha$ and $\beta$. Since $C$ contains no conditional dependencies changing the value of one attribute does not affect the preference order over another attribute's domain. This means the only way that a worsening flipping sequence exists is if for each attribute $\alpha$ has a value that is at least as preferred as the value in $\beta$. This argument also works for a worsening flipping sequence from $\beta$ to $\alpha$.

Therefore, the only time $\alpha \bowtie_C \beta$ is if for at least one attribute $v_i$, $\alpha \succ_{v_i} \beta$ and for at least one attribute $v_j$, $\beta \succ_{v_j} \alpha$. Since all attributes have the same rank by the definition of dominance in a CLPM we must have $\alpha \bowtie_\pi \beta$ as well. Thus, the order induced by $C$ is identical to the order induced by $\pi$. $\square$

We are able to further generalize these results to show how approximation accuracy is affected by disjoint subgraphs, which we call independent subnetworks, in a CP-net's dependency graph. We formally define such independent subnetworks as follows:

**Definition 15** (Independent Subnetwork). *Given an acyclic CP-net $C = (\mathcal{V}, E, T)$, the set $V \subset \mathcal{V}$ is an independent subnetwork of $C$ if for all $v \in V$ there does not exist $w \in (\mathcal{V} \setminus V)$ such that $(v, w) \in E$ or $(w, v) \in E$*

We can view an independent subnetwork as its own local CP-net where alternatives are projected down to only the attributes contained in the subnetwork. Using these projections we can define a subnetwork's preference order, see Definition 16.

Using the independent subnetwork orders we can better predict the accuracy of a CLPM approximation of the entire CP-net.

**Definition 16** (Independent Subnetwork Dominance)**.** *Given an acyclic CP-net $C = (\mathcal{V}, E, T)$, an independent subnetwork $V$, and two alternatives $\alpha$ and $\beta$, $\alpha \succ_V \beta$ if there exists a flipping sequence from $\alpha[V]$ to $\beta[V]$ which does not change any values in $\mathcal{V} \setminus V$.*

For any independent subnetwork of a CP-net we show that there exists an approximating CLPM, for the entire CP-net, which will induce the local dominance relations of the subnetwork, Lemma 31.

**Lemma 31.** *Given an acyclic CP-net $C = (\mathcal{V}, E, T)$, an independent subnetwork $V$, and two alternatives $\alpha$ and $\beta$, if $\alpha \succ_V \beta$ then there exists an approximating CLPM, $\pi = (C, r)$ where $\alpha \succ_\pi \beta$.*

*Proof.* By lemma 28 we know that if $\alpha$ and $\beta$ differ by a single attribute then any approximating CLPM will induce the relations given by a single flip. We also know that a CLPM's order is transitive and thus will induce relations induced by a flipping sequence of any length. We also know that since $V$ is an independent subnetwork that there exists no edge to a member of $V$ from a member of $\mathcal{V} \setminus V$ there exists a consistent ranking $r$ such that for any member $s \in V$ and any member $v \in \mathcal{V} \setminus V$, $r(s) < r(v)$. Given that $\alpha \succ_S \beta$ we know that there must exist some attribute $s \in V$ such that $\alpha[s] \succ \beta[s]$ and thus by the definition of dominance for a CLPM dominance will be decided in the ranks which contain members of $V$ using ranking $r$. This means that the CLPM $\pi = (C, r)$ attributes in $\mathcal{V} \setminus V$ will not be considered. Thus there exists a CLPM such that $\alpha \succ \beta$. □

Applying this knowledge requires that we first know how the independent subnetworks interact in the context of the greater CP-net. When two independent subnetworks disagree about the ordering of two alternatives those alternatives are considered incomparable wrt the full CP-net, see Proposition 5. Corollary 4 shows that the aggregated CLPM approximation also renders such pairs incomparable.

**Proposition 5.** *Given an acyclic CP-net $C = (\mathcal{V}, E, T)$, two independent subnetworks of $C$, $V$ and $W$, and two alternatives $\alpha$ and $\beta$ if $\alpha \succ_V \beta$ and $\beta \succ_W \alpha$ then $\alpha \bowtie_{CLPM(C)} \beta$.*

*Proof.* By lemma 31 we know that there exists at least two CLPMs where one induces $\alpha \succ \beta$ and the other induces $\beta \succ \alpha$. By theorem 2 we know that there must exist a CLPM $\pi = (C, r)$ such that $\alpha \bowtie_\pi \beta$ and thus, by the definition of $\succ_{CLPM(C)}$ we know that $\alpha \bowtie_{CLPM(C)} \beta$. □

**Corollary 4.** *Given an acyclic CP-net $C = (\mathcal{V}, E, T)$, two independent subnetwork of $C$, $V$ and $W$, and two alternatives $\alpha$ and $\beta$, if $\alpha \succ_V \beta$ and $\beta \succ_W \alpha$ then $\alpha \bowtie_C \beta$.*

*Proof.* Given proposition 5, Corollary 1, and the definition of the aggregated lexicographic evaluation if $\alpha \succ_V \beta$ and $\beta \succ_W \alpha$ then $\alpha \bowtie_{CLPM(C)} \beta$ and thus $\alpha \bowtie_C \beta$. □

Corollary 4 allows us to state that when two independent subnetworks disagree the aggregated CLPM approximation will render the affected pair of alternatives incomparable. This alone shows that CP-nets whose dependency graph can be divided into independent subnetworks have greater guaranteed accuracy than CP-nets whose dependency graphs cannot. Additionally, if there exists an approximating CLPM for each independent subnetwork, which accurately approximates it, then the aggregated CLPM approximation will approximate the CP-net's preference order with no errors, Theorem 5.

**Theorem 5.** *Given a CP-net $C = (\mathcal{V}, E, T)$ which can be decomposed into independent subnetworks $C_1, C_2, \ldots, C_k$, such that for each $C_i$, $1 \leq i \leq k$, $\succ_{C_i}$ is a total order, $\alpha \succ_C \beta$ iff $\alpha \succ_{CLPM(C)} \beta$.*

*Proof. If $\alpha \succ_C \beta$ then $\alpha \succ_{CLPM(C)} \beta$.*

By Theorem 1 we know that this is true since all approximating CLPMs will induce dominance if it exists in the original CP-net and thus will be induced by the aggregated CLPM approximation.

*If $\alpha \succ_{CLPM(C)} \beta$ then $\alpha \succ_C \beta$.*

Suppose not and $\alpha \succ_{CLPM(C)} \beta$, but $\alpha \not\succ_C \beta$. By Theorem 1 we know that this could only occur if $\beta \bowtie_C \beta$. Since each subnetwork constructs a total order the means there must exist some $C_i$ and $C_j$ such that $\alpha \succ_{C_i} \beta$ and $\beta \succ_{C_j} \alpha$. By Corollary 4 this means that $\alpha \succ_{CLPM(C)} \beta$, thus a contradiction. □

These results give us insight into the structures which better accommodate an aggregated CLPM approximation and give us some settings where we can apply a single CLPM to accurately approximate a CP-net's order.

## 4.5 Case Study Discussion

As a case study on the limitation of subsumption, the CLPM approximation of CP-nets is valuable. Perhaps the largest contribution in this sense is that there exist many CP-nets which can be represented by CLPMs. For establishing subsumption we only care about the ability to convert models to equivalent models, i.e. if one model cannot be converted into another language then subsumption is not derived. This means that two languages can share a great deal of their expressive power, but unless one is completely contained inside the other, subsumption will fail to derive such a relation.

CLPMs are capable of closely approximating CP-nets. This reduces the computational complexity of extracting certain types of information, namely dominance, at the cost of accuracy. Given the closeness of CLPMs and CP-nets, without a subsumptive relation, this case study shows the inability of subsumption to account for more nuanced relations between languages. This reinforces our belief that subsumptive analysis, while powerful, is only a part of any more rigorous analysis of preference representation languages.

# Chapter 5 Developed Preference Learning Techniques

Learning algorithms exist for a variety of languages, including LPMs [Schmitt and Martignon, 2006], LP-trees [Bräuning and Hüllermeier, 2016], and tree structured CP-nets [Allen et al., 2017], but most of these do not guarantee that the learned model is optimal. This is due to many of these learning problems being computationally difficult.

This complexity issue is exemplified by both LPMs and CP-nets. In both cases, the problem of learning a model which most accurately satisfies a set of examples is NP-hard [Alanazi et al., 2016, Schmitt and Martignon, 1999]. In the case of LPMs, it is possible to approximate the optimal model using a greedy algorithm which minimizes the number of incorrectly decided examples at each step. The model built using this algorithm is guaranteed to correctly decide at least half of the examples [Schmitt and Martignon, 2006]. In the case of CP-nets there has been some success using local search techniques to learn tree structured CP-nets [Allen et al., 2017].

We propose two methods of learning preferences from a set of examples (pairwise comparisons) provided by an agent. The first is an application of simulated annealing applied to the problem of learning RPFs and PTs. The second is applying the well studied (elsewhere) machine learning technique of artificial neural networks (ANNs). Both of these techniques are based on metaheuristics and do not come with theoretical performance guarantees. Despite this, we provide data that indicates both techniques are capable of accurately learning qualitative preferences.

## 5.1 Simulated Annealing and Logical Preferences

The goal of the preference learning task is to construct a preference model based on a collection of *pairwise comparison examples.* Each pairwise comparison example is a triple $(\alpha, \beta, R)$ where $\alpha$ and $\beta$ are alternatives and $R \in \{\succ, \succeq, \approx, \preceq, \prec\}$ is the preference relation between them. For example $(01, 11, \succ)$ means an agent strictly prefers 01 to 11. In settings where examples come from more than one agent we will append the agent's identifier to the example, thus in multi-agent settings examples are 4-tuples rather than triples.

When learning RPFs and PTs we start by considering the domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$ and its related language of propositional logic $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$. For each value $j$ of an attribute $v_i$ the language has a propositional atom $x_{i,j}$. This atom's intended reading is: an alternative has value $j$ on attribute $v_i$. Thus, the language $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$ contains one atom for each attribute-value pair, rather than each attribute (as occurs in the binary combinatorial domain setting). Similarly, there is a one-to-one correspondence between alternatives in $\mathcal{C}(\mathcal{V}, \mathcal{D})$ and some truth assignments to atomic formulas in the language $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$. The formulas of the language are formed in the standard way from atoms by means of the logical connectives $\wedge, \vee,$ and $\neg$. Often is it convenient to refer to a particular atom or its negation, we call this a literal. Formulas from $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$ can be understood as expressing desirable properties of alternatives. The number of atoms is defined by the following expression:

$$\sum_{v_i \in \mathcal{V}} |\mathcal{D}(v_i)|.$$

In the setting where all attributes have the same domain size $m$, the number of atoms in the language $L(\mathcal{C}(\mathcal{V}, \mathcal{D}))$ is $n * m$, where $n$ is the number of attributes. Thus, the number of atoms is polynomial in the size of the domain, which allows us to build compact preference representations, assuming we limit the size of the formulas we build. It is due to this reduction that we are able focus on combinatorial domains with binary attribute domains without loss of generality.

We formally state the problem of preference model learning, or PML for short:

**Problem 4** (Preference Model Learning PML($\mathcal{L}$)). *We assume a fixed preference* *language $\mathcal{L}$ for combinatorial domains over binary attributes. Given an integer $k$ and a set of examples $\varepsilon$ representing an agent's choices over alternatives from $\Omega$, decide whether there is a preference model $M$ in $\mathcal{L}$ such that the model satisfies (decides in the same way) at least $k$ examples from $\varepsilon$.*

This problem definition represents the decision problem related to learning a particular preference model. In this work, we will focus on the case when $\mathcal{L} \in \{\text{RPF}, \text{PT}\}$.

Before we discuss the complexity of solving PML for both RPFs and PTs, we must discuss their expressive limitations. As shown above RPFs can express any total preorder, while PTs can express any preorder. This indicates the limit of their expressive power, but not how large those models may need to be. Some (total) preorders can only be represented by models of exponential size, wrt the number of attributes in the domain. When we learn models from examples, another, more practical bound involves the size of the example set.

**Lemma 32.** *For any consistent set of examples $\varepsilon$ over a binary combinatorial domain $\mathcal{C}(\mathcal{V})$ that contains only $\succ, \prec, \approx$ relations, there exists an RPF which satisfies all relations in $\varepsilon$ and is of size polynomial in the size of $\varepsilon$.*

*Proof.* Since alternatives are vectors of attribute value pairs we can describe a formula $\varphi_\alpha$ for each alternative $\alpha$ which is only satisfied by that alternative. The formulas are defined as $\varphi_\alpha \equiv \wedge_{v \in \mathcal{V}} x_{v,\alpha[v]}$. Since $\varepsilon$ is consistent there is a way to sort the alternatives such that no example is violated. Using this sorting we build a ranking function $r$ over the alternatives in $\varepsilon$ such that higher ranked alternatives appear later in the sorted list. For each rank $i$ we build an equivalent $\varphi_i = \vee_{r(\alpha)=i} \varphi_\alpha$, thus $\varphi_i$ is only satisfied by alternatives with rank $i$ according to $r$. These formulas are collected into an RPF $\boldsymbol{\varphi}$. The ranking function $\boldsymbol{\varphi}$ induces is identical to the ranking function $r$ which orders the alternatives consistently with $\varepsilon$. Thus there exists an RPF which satisfies all relations in $\varepsilon$. Since there are up to $2 * |\varepsilon|$ alternatives to consider and each alternative's formula is used only once, with a size identical in length to the size of the alternative, the size of the built PF is $\mathcal{O}(|\varepsilon|)$. $\square$

**Lemma 33.** *For any consistent set of examples $\varepsilon$ over a binary combinatorial domain $\mathcal{C}(\mathcal{V})$ there exists a PT which satisfies all relations in $\varepsilon$ and is of size polynomial in the size of $\varepsilon$.*

*Proof.* By Lemma 10 we know that if we can build a set of satisfaction vectors $S$, with one entry for each alternative in $\mathcal{C}(\mathcal{V})$, such that $s(\alpha) \geq_{\text{Pareto}} s(\beta)$ if $\alpha \succeq \beta$, then there exists a preference theory which induces $\succeq$ as its associated preorder. For our purposes we will consider the alternatives in $\mathcal{C}(\mathcal{V}, \mathcal{D})$ to be split into clusters by the equivalence classes defined by the $\approx$ derived from $\succeq$. When we refer to alternatives below we really mean an equivalence class of alternatives.

We begin our construction by setting the first element of each alternative's satisfaction vector to its canonical ranking as defined by the canonical ranking function, $r_{c,\succeq}$. This means that, just considering the first element, all dominance relations in $\succeq$ are satisfied. This does not hold for incomparable alternatives relations, but as long as inserting incomparabilities does not remove these dominance satisfying properties we only need to worry about pairs of alternative $(\alpha, \beta)$ such that $\alpha \bowtie \beta$.

For each incomparable pair $(\alpha, \beta)$ induced by $\succeq$ we follow the process detailed below:

- Append $1, 2$ to the satisfaction vector for $\alpha$.
- Append $2, 1$ to the satisfaction vector for $\beta$.
- Append $1, 1$ to the satisfaction vector for any alternative $\gamma$ such that $r_{c,\succeq}(\gamma) < r_{c,\succeq}(\alpha)$
- Append $2, 2$ to the satisfaction vector for any alternative $\gamma$ such that $r_{c,\succeq}(\gamma) > r_{c,\succeq}(\beta)$

- Append $1, 1$ to the satisfaction vector for any alternative $\gamma$ such that $r_c c, \succeq(\alpha) < r_{c,\succeq}(\gamma) < r_{c,\succeq}(\beta)$ and $\alpha \not\bowtie \gamma$.
- Append $2, 2$ to the satisfaction vector for any alternative $\gamma$ such that $r_c c, \succeq(\alpha) < r_{c,\succeq}(\gamma) < r_{c,\succeq}(\beta)$ and $\alpha \bowtie \gamma$.

For this construction we assume, without loss of generality, that $r_{c,\succeq}(\alpha) < r_{c,\succeq}(\beta)$.

Using this construction any two incomparable alternatives now have Pareto incomparable satisfaction vectors, since they each have at least one element strictly better than the other. Importantly, if there is a more preferred alternative than the pair each added element is at least as preferred as the ones added to the pair, thus preserving the dominance relation. Similar can be said for alternatives which are less preferred than the pair.

For alternatives which have a canonical rank in between those of the pair we know, by Lemma 12, they must be incomparable with at least one of the alternatives in the pair. The elements added to the satisfaction vectors of "in between" alternatives are such that dominance will still be satisfied after they are added.

Thus the constructed set of satisfaction vectors is such that $s(\alpha) \geq_{\text{Pareto}} s(\beta)$ if $\alpha \succeq \beta$ holds. This means that for any arbitrary preorder $\succeq$ there is a preference theory $P$ such that for any pair of alternatives $\alpha, \beta \in \mathcal{C}(\mathcal{V}, \mathcal{D})$ if $\alpha \succeq \beta$ then $\alpha \succeq_P \beta$.

For each member of the satisfaction vector there must exist some RPF which induces the appropriate ranking function for that member. It is clear that each member defines a total order over the alternatives in $\varepsilon$. As shown in Lemma 32 these RPFs are of size polynomial in the size of $\varepsilon$. The total number of RPFs needed to construct the PT which satisfies all examples in $\varepsilon$ is at most $2 * |\varepsilon| + 1$ since we need two additional elements in the satisfaction vector per incomparable example in $\varepsilon$ and one RPF for the canonical order element. This means that the PT if of size $\mathcal{O}(|\varepsilon| * (2 * |\varepsilon| + 1))$ which is equivalent to $\mathcal{O}(|\varepsilon|^2)$ and thus is of size polynomial wrt to the size of the example set. □

Lemmas 32 and 33 show that it is possible to construct models which replicate consistent example sets that are of polynomial size wrt the example sets. These results are encouraging from a computational complexity standpoint. Unfortunately, PML is NP-complete for both RPFs and PTs, see Theorems 6 and 7.

**Theorem 6.** *PML(RPF) is NP-complete.*

*Proof.* (*Membership.*)
If there is an RPF satisfying at least $k$ of the examples in $\varepsilon$, there is an RPF of size polynomial in $k$ satisfying those $k$ examples, by Lemma 32. That RPF has size polynomial in the size of $\varepsilon$ and satisfies at least $k$ examples in $\varepsilon$. Therefore, each YES instance of the problem has a witness of size polynomial in the size of $\epsilon$. Thus, membership follows.

(*Hardness.*)
The problem of feedback arc set consists of an instance $(G, l)$ where $G$ is a graph and $l$ is a non-negative integer. An instance is a member of feedback arc set if there is a way to remove at most $l$ edges from $G$ such that $G$ becomes acyclic. Given an instance $I = (G = (V, E), l)$ of feedback arc set we define $f(I)$ to be an instance of PML(RPF) $(k, \varepsilon)$. We define the combinatorial domain of alternatives for $f(I)$ by setting $\mathcal{V} = \{v|v \in V\}$ and for each $v \in \mathcal{V}$, $\mathcal{D}(v) = \{0, 1\}$. We further define an alternative $\alpha_v$ for each vertex $v \in V$ such $\alpha_v$ is defined by setting $\alpha_v[v] = 1$ and for all $w \in V, w \neq v, \alpha_v[w] = 0$. We build the example set $\varepsilon = \{\alpha_u \succ \alpha_v|(u, v) \in E\}$. We set $k$ in $f(I)$ to $|E| - l$.

- *If $I \in FAS$ then $f(I) \in PML(RPF)$*
  If $I \in FVS$ then there exists a graph $G' = (V', E')$ such that $V' \equiv V$, $E' \subseteq E$ and $G'$ is acyclic. This means we can topologically sort the vertices in $G'$ such that for any edge $(u, v) \in E'$ vertex $u$ comes before vertex $v$. We denote the

first element of the sort to be $V_1$, the second $V_2$, and so on. This translates to a linear order over the alternatives in $\varepsilon$. Since each vertex's corresponding alternative is indicated by a value of 1 for a single unique attribute each $\varphi \in \boldsymbol{\varphi}$ needs only a single literal. The set $\boldsymbol{\varphi}$ is built by setting the literal in $\varphi_i$ is equal to $x_{V_i,1}$. This means that each edge $(u, v) \in E'$ we satisfy $\alpha_u \succ_{\boldsymbol{\varphi}} \alpha_v$ since $\alpha_u$ is the only alternative in $\varepsilon$ which can satisfy $x_{u,1}$, similarly for $v$, and because $\boldsymbol{\varphi}$ conforms to the topological sort the formula consisting of $x_{u,1}$ must come before the one consisting of $x_{v,1}$. Thus for each edge which remains in $E'$ we satisfy its equivalent example in $\varepsilon$ meaning we satisfy at least $|E| - l = k$ examples making $f(I)$ a valid instance of PML(RPF).

- *If $I \notin$ FAS then $f(I) \notin$ PML(RPF)*

  Given an instance $I = (G = (V, E), l)$ such that $I \notin$ FVS we know that it is impossible to remove up to $l$ vertices and change $G$ into an acyclic graph. This means that for all possible graphs $G' = (V', E')$ such that $V = V'$, $E' \subseteq E$, and $|E'| \geq |E| - l$ there exists a cycle in $G'$. This translates into a cyclical relationship among the examples in $\varepsilon$ such that we would have to remove more that $l$ examples so that the resultant example set is consistent. Since an RPF always induces a consistent total preorder, an RPF that satisfies at least $k = |E| - l$ examples cannot exist.

□

**Theorem 7.** *PML(PT) is NP-complete.*

*Proof.* (*Membership.*)
If there is a PT satisfying at least $k$ of the examples in $\varepsilon$, there is a PT of size polynomial in $k$ satisfying those $k$ examples, by Lemma 33. That PT has size polynomial in the size of $\varepsilon$ and satisfies at least $k$ examples in $\varepsilon$. Therefore, each YES instance of the problem has a witness of size polynomial in the size of $\epsilon$. Thus, membership

follows.

(*Hardness.*)

The problem of feedback arc set consists of an instance $(G, l)$ where $G$ is a graph and $l$ is a non-negative integer. An instance is a member of feedback arc set if there is a way to remove at most $l$ edges from $G$ such that $G$ becomes acyclic. Given an instance $I = (G = (V, E), l)$ of feedback arc set we define $f(I)$ to be an instance of PML(PT) $(k, \varepsilon)$. We define the combinatorial domain of alternatives for $f(I)$ to be $\mathcal{V} = \{v | v \in V\}$ and for each $v \in \mathcal{V}$, $\mathcal{D}(v) = \{0, 1\}$. We further define an alternative $\alpha_v$ for each vertex $v \in V$ such $\alpha_v$ is defined by setting $\alpha_v[v] = 1$ and $\forall w \in V, w \neq v, \alpha_v[w] = 0$. We build the example set $\varepsilon = \{\alpha_u \succ \alpha_v | (u, v) \in E\}$. We set $k$ in $f(I)$ to $|E| - l$.

- *If $I \in$ FAS then $f(I) \in$ PML(PT)*

  If $I \in FVS$ then there exists a graph $G' = (V', E')$ such that $V' \equiv V$, $E' \subseteq E$ and $G'$ is acyclic. This means we can topologically sort the vertices in $G'$ such that for any edge $(u, v) \in E'$ vertex $u$ comes before vertex $v$. We denote the first element of the sort to be $V_1$, the second $V_2$, and so on. This translates to a linear order over the alternatives in $\varepsilon$. Since each vertex's corresponding alternative is indicated by a value of 1 for a single unique attribute each $\varphi \in \boldsymbol{\varphi}$ needs only a single literal. $\boldsymbol{\varphi}$ is built such that the literal in $\varphi_i$ is equal to $x_{V_i,1}$. This means that for each edge $(u, v) \in E'$, $\alpha_u \succ_{\boldsymbol{\varphi}} \alpha_v$ since $\alpha_u$ is the only alternative in $\varepsilon$ which can satisfy $x_{u,1}$, similarly for $v$, and because $\boldsymbol{\varphi}$ conforms to the topological sort the formula consisting of $x_{u,1}$ must come before the one consisting of $x_{v,1}$. Thus for each edge which remains in $E'$ we satisfy its equivalent example in $\varepsilon$ meaning we satisfy at least $|E| - l = k$ examples making $f(I)$ a valid instance of PML(PT).

- *If $I \notin$ FAS then $f(I) \notin$ PML(PT)*

Given an instance $I = (G = (V, E), l)$ such that $I \notin$ FVS we know that it is impossible to remove up to $l$ vertices and change $G$ into an acyclic graph. This means that for all possible graphs $G' = (V', E')$ such that $V = V'$, $E' \subseteq E$, and $|E'| \geq |E| - l$ there exists a cycle in $G'$. This translates into a cyclical relationship among the examples in $\varepsilon$ such that we would have to remove more that $l$ examples so that the resultant example set is consistent. Since a ranked PT always induces a consistent preorder there cannot exists a ranked PT that satisfies at least $k = |E| - l$ examples.

□

Since the problem of learning an RPF or PT maximizing the number of satisfied examples is NP-hard there is no efficient way to build such models, assuming $P \neq NP$. In order to learn RPF and PT models in a reasonable amount of time we must rely on building approximations of the optimal models. We study an approach based on simulated annealing [Kirkpatrick et al., 1983]. A generic version of simulated annealing is given in Algorithm 2.

**Algorithm 2** Basic Simulated Annealing

```
procedure SIMULATED ANNEALING($t_0$: the initial temperature)
    result ← RANDOM-MODEL()
    $t \leftarrow t_0$
    while Stopping Criterion not Met do
        temp ← RANDOM-NEIGHBOR(result)
        if EVAL(temp) is better than EVAL(result) then
            result ← temp
        else
            $\Delta$ ← EVAL(result) - EVAL(temp)
            Replace result with temp with probability $e^{\frac{-\Delta}{t}}$
        end if
        Reduce $t$
    end while
end procedure
```

Before simulated annealing can be applied we need a method for representing

RPFs and PTs which has a well defined neighbor function. The representation we use makes the assumption that our logical formulas are in disjunctive normal form and that the overall size of the model we learn is static. Limiting the models in this manner allows us to define an RPF (resp. PT) model by a string of literals (an atom or their negation) which makes defining the neighbor relation simple. We detail our representation below.

Our simulated annealing RPF and PT representation assumes that each formula is in *disjunctive normal form* (DNF). A DNF formula is a disjunction of conjunctions of literals. A DNF formula's size is specified by a pair $(i, j)$. A $i, j$-DNF formula is a disjunction of $i$ conjunctions with each conjunction consisting of $j$ literals. Given this convention, a $i, j$-DNF formula can be expressed by a string of literals of length $i * j$. DNF is not the only normal form, but it makes sense from a linguistic point of view. For example a person would tend to express their desired pizza by saying: "mushrooms and sausage or olives and pepperoni." In this case we have two statements "mushroom and sausage" and "olives and pepperoni" which describe desirable pizzas. Each consists of a conjunction of two atoms and the statements form a single disjunction, thus it is a $1, 2$-DNF formula.

All the formulas we consider in this work are in DNF. Of course, our logical preference representations RPFs and PTs use collections of formulas and so require additional specification. We specify RPFs using a type triple $(i, j, k)$ which means that there are $k$ formulas, which are $i, j$-DNF formulas. So the RPF $\boldsymbol{\varphi} = ((x_{1,2} \vee x_{3,2}), (x_{3,2} \vee x_{1,1}), (x_{1,4} \vee x_{3,3}))$ is a $(2, 1, 3)$-RPF. Expanding this to PTs we prepend two elements $a, b$ where $a$ is the number of ranks, and $b$ is the number of RPFs per rank.

We define two RPFs (respectively, PTs) to be neighbors if their literal string representations only differ on a single literal. Using our representation and neighbor definition we can adapt simulated annealing to our learning problem. In order to increase the accuracy of our simulated annealing algorithm we run a naive hill climbing algorithm on the candidate solution produced by the simulated annealing process. Hill climbing searches all neighbors of a candidate solution and picks the one which improves accuracy the most, and repeats until no improving neighbor exists. If no improving neighbor exists it returns the candidate solution. Adding this hill climbing algorithm to our learning process guarantees that the learned model is a local optimum.

The simulated annealing algorithm requires that we set some hyperparameters before execution. As we have implemented it, the hyperparameters are the cooling function, temperature stopping criterion, and initial temperature. We use a cooling function with an exponential cooling rate. Specifically, we divide the current temperature by $1 + 10^{-3}$ at each step. Our stopping criterion is when the temperature is below $10^{-7}$. We chose an initial temperature of 100. This gives our simulated annealing a total of $20,734$ iterations before completion. These hyperparameters were chosen through experimentation and do not reflect a concerted effort to find optimal settings for this problem, merely ones which performed well. Again, we are dealing with approximations and so it may be possible to further increase performance through hyperparameter tuning.

We compare our learning results for RPFs and PTs against a greedy algorithm for learning LPMs from the literature [Schmitt and Martignon, 2006]. This algorithm guarantees at least half of all training examples are satisfied, assuming all examples have either $\succ$ or $\prec$ as their relation. Should the example set be completely satisfiable by an LPM, the algorithm will output such an LPM. This greedy algorithm represents the baseline we compare our developed learning techniques against.

Most languages can be divided into sublanguages based on restrictions placed over properties of their models. For example, CP-nets which have a bound on the maximum indegree in their dependency graphs represent a sublanguage of the wider

CP-net language. A particular sublanguage may produce different preorders than another sublanguage from the same language. This means that how we generate random models of a language dictates the preorders that are produced. With this understanding in mind we identify different generating parameters of CP-nets and CLPMs using a numbering scheme such that a $k$-CP-net means a CP-net whose dependency graph has an indegree of at most $k$. Similarly, a $k$-CLPM is a CLPM whose attributes are conditioned on at most $k$ other attributes. For LP-trees we attach two numbers, the number of times the tree branches on any given path through the tree, i.e. the degree of conditional importance, and the bound on the number of conditioning attributes for any one attribute, i.e. the degree of conditional preference. This means a $2,3$-LP-tree will branch two times on any given path through the tree and any attribute is at most conditioned on three other attributes. Most of these random models were generated using naive methods with no explicit bias being placed on their generation to enusre uniform generation or other potentially desirable properties. The exception to this is the generation of CP-nets which was achieved using a software from Allen et al. [2016] which generates acylcic CP-nets uniformly at random.

Table 5.1: Simulated annealing learning accuracy in single-agent settings.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 1.00 | 0.66 | 0.92 | 0.97 | 0.75 |
| 1,1,3-RPF | 0.64 | 1.00 | 1.00 | 0.72 | 0.85 |
| 2,2,7-RPF | 0.61 | 0.73 | 0.99 | 0.70 | 0.66 |
| 3,3,2,2,7-PT | 0.47 | 0.32 | 0.45 | 0.89 | 0.72 |
| 1,3,1,1,3-PT | 0.51 | 0.44 | 0.59 | 0.94 | 1.00 |
| 0-CP-net | 0.19 | 0.16 | 0.21 | 0.96 | 0.80 |
| 7-CP-net | 0.73 | 0.54 | 0.75 | 0.94 | 0.75 |
| 0-CLPM | 0.53 | 0.40 | 0.40 | 0.96 | 0.85 |
| 7-CLPM | 0.53 | 0.35 | 0.56 | 0.97 | 0.86 |
| 1,1-LP-tree | 0.92 | 0.67 | 0.92 | 0.98 | 0.74 |
| 3,3-LP-tree | 0.91 | 0.68 | 0.91 | 0.97 | 0.72 |

Our example sets were built by generating a model, at random, from a given language and then selecting a set of 100 random pairs of alternatives from a combinatorial domain consisting of eight binary attributes. Using the model, we determined the preference relations between them and thus produced 100 examples. In order to test how well our learned models work with unseen examples we used a five-fold cross validation scheme where we removed 1/5 of the examples (20) from the example set and trained our models on the other 80 examples. Each instance was run 25 times, which due to our five-fold cross validation strategy means 125 models were trained and their accuracies averaged. Table 5.1 shows the proportion of examples satisfied for the training example sets. Table 5.2 shows the proportion of the unseen examples that were satisfied.

Table 5.2: Simulated annealing validation accuracy in single-agent settings.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.98 | 0.60 | 0.75 | 0.80 | 0.61 |
| 1,1,3-RPF | 0.62 | 1.00 | 0.98 | 0.62 | 0.84 |
| 2,2,7-RPF | 0.55 | 0.68 | 0.94 | 0.54 | 0.58 |
| 3,3,2,2,7-PT | 0.40 | 0.27 | 0.32 | 0.59 | 0.57 |
| 1,3,1,1,3-PT | 0.49 | 0.40 | 0.48 | 0.74 | 0.98 |
| 0-CP-net | 0.18 | 0.10 | 0.13 | 0.69 | 0.63 |
| 7-CP-net | 0.65 | 0.49 | 0.63 | 0.71 | 0.63 |
| 0-CLPM | 0.51 | 0.35 | 0.29 | 0.71 | 0.73 |
| 7-CLPM | 0.50 | 0.29 | 0.45 | 0.74 | 0.76 |
| 1,1-LP-tree | 0.84 | 0.62 | 0.75 | 0.78 | 0.60 |
| 3,3-LP-tree | 0.82 | 0.63 | 0.76 | 0.79 | 0.60 |

As we expected the average accuracy over the validation set is less than over the training set. The difference between the two averages is less than 0.20 in all settings, except for $3,3,2,2,7$-PTs learning from $3,3,2,2,7$-PT example sets, indicating at least some level of generalization in both the LPM baseline and simulated annealing. We suspect that the poor generalization performance of $3,3,2,2,7$-PTs learning from $3,3,2,2,7$-PT example sets is due to such models generating preorders which are harder to predict from a small number of examples. We also note that the poor accuracy of LPMs and RPFs when attempting to approximate orders induced by CLPMs and CP-nets is most likely due to the presence of examples with incomparable alternatives. Since LPMs and RPFs induce total orders and total preorders, respectively, they are incapable of reproducing those examples, meaning they will always have reduced accuracy. Moreover, CP-nets with no conditional preferences have the highest density of incomparable pairs of alternatives and LPMs and RPFs worst accuracy occurred on those example sets. There is some variability to the accuracy of these approximations based on the exact example sets they learn from. In order to study how different the accuracy can be from case to case we computed the standard deviation for the validation set accuracy, see Table 5.3.

Table 5.3: Simulated annealing validation accuracy standard deviation in single-agent settings.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.04 | 0.12 | 0.10 | 0.08 | 0.11 |
| 1,1,3-RPF | 0.12 | 0.01 | 0.04 | 0.13 | 0.11 |
| 2,2,7-RPF | 0.13 | 0.15 | 0.06 | 0.13 | 0.16 |
| 3,3,2,2,7-PT | 0.14 | 0.11 | 0.10 | 0.13 | 0.12 |
| 1,3,1,1,3-PT | 0.16 | 0.14 | 0.13 | 0.13 | 0.04 |
| 0-CP-net | 0.08 | 0.06 | 0.08 | 0.11 | 0.10 |
| 7-CP-net | 0.12 | 0.15 | 0.14 | 0.13 | 0.14 |
| 0-CLPM | 0.22 | 0.21 | 0.18 | 0.12 | 0.14 |
| 7-CLPM | 0.24 | 0.18 | 0.19 | 0.12 | 0.11 |
| 1,1-LP-tree | 0.09 | 0.12 | 0.11 | 0.11 | 0.12 |
| 3,3-LP-tree | 0.09 | 0.12 | 0.11 | 0.10 | 0.11 |

The standard deviation appears to vary wrt the language of the model producing the example set rather than the language of the model learning from the example set. This implies that the variance of how accurate the approximations are is more closely tied to the generating model's language. Looking more closely at these relations we see that more expressive languages, such as PTs and CP-nets, tend to include more variance, which one expects given that the set of possible preference orderings they can produce is larger.

These tests assume that one is learning from an example set generated by a single agent. There are settings where one might wish to make a group decision, or to learn the consensus preferences of a group. The task of learning preferences for a group is one that simulated annealing can easily be adapted to. Before, we considered the proportion of satisfied examples to be the measure of how well a candidate solution performed. There are two main ways to measure performance in multi-agent settings: utilitarian and maximin, see Problem 5.

**Problem 5** (Rank Aggregation Model Learning RAML($\mathcal{L}$))**.** *We assume a fixed preference language $\mathcal{L}$ for a combinatorial domain $\mathcal{C}(\mathcal{V})$ with binary attributes. Given an integer $k$ and a collection of sets of examples $(\varepsilon_1, \ldots, \varepsilon_n)$, each representing choices over alternatives from $\mathcal{C}(\mathcal{V})$ of a different agent in a group of $n$, decide whether there is a preference model $M$ in $\mathcal{L}$ such that the model satisfies (decides in the same way) at least $k$ examples in $\varepsilon = \bigcup_{i=1}^{n} \varepsilon_i$ (*utilitarian *version of the problem) or in each set $\varepsilon_i$ (*maximin *version of the problem).*

When aggregating preferences the topic of fairness becomes pertinent. Fairness can be quantified in many different ways. We have chosen utilitarian and maximin as our criteria, and we now consider how they can be justified wrt fairness. In the case of the utilitarian criterion the fairness justification is that by maximizing overall satisfaction we maximize the satisfaction of the individual agents. This is not always the case. For example, if we have two agents, each with 100 examples, and one agent gets all their examples satisfied and the other gets none satisfied then the utilitarian criterion considers this the same as both agents having 50 examples satisfied, since 100 examples total are satisfied in both scenarios. Most people would consider the second situation, an even split of satisfied examples, fairer than the first. The maximin criterion makes a distinction between the two cases given that in the first the least satisfied agent has 0 examples satisfied. According to maximin the second situation is preferable to the first.

By optimizing for the maximin criterion we attempt to make the agent who is worst off as well off as possible. This means that if we see an increase in accuracy under maximin then all agents must now be satisfied above that level, although individual scores may fluctuate up or down. The maximin criterion supports fairness by attempting to find an aggregation where everyone is as satisfied as possible without potentially disregarding any agent's preferences. We show that the problem of RAML($\mathcal{L}$) is NP-hard for both settings by reducing from the single agent case. NP-completeness for both problems is shown in Corollary 5.

**Corollary 5.** *The problem RAML($\mathcal{L}$) (in each of its versions) is NP-complete for $\mathcal{L}$ = PF, PT and RPT, even when formulas appearing in preference formulas are in DNF.*

*Proof.* (*Membership.*)
Membership in NP is shown identically to the single-agent setting because any collection of examples representable by a PF, PT, or RPT must be consistent with a preorder (in the case of PFs a total preorder). This means that if a PF, PT, or RPT does satisfy RAML then there exists a PF, PT, or RPT (resp.) which is polynomial in size wrt to set of consistent examples and so checking to see if the guessed PF, PT, or RPT satisfies the condition takes polynomial time, wrt the size of the example set.

(*Hardness.*)
We show hardness by reducing from PML($\mathcal{L}$) to RAML($\mathcal{L}$). Given an instance $I = (\varepsilon, k)$ of PML($\mathcal{L}$) we build a mapping function $f(I) = (\varepsilon', k')$ such that $\varepsilon'$ contains all examples in $\varepsilon$ where each example has the agent identifier 1 appended. The value $k'$ is set to be equal to $k$.

- *If $I \in PML$ then $f(I) \in RAML$*

  Given an instance of PML($\mathcal{L}$) if there exists a witness model $M \in \mathcal{L}$ such that the instance $I = (\varepsilon, k)$ is a YES instance than it is simple to see that $M$ is also a witness model for $f(I)$ in RAML making $f(I)$ a YES instance as well. If there exists no model

- *If $I \notin PML$ then $f(I) \notin RAML$*

  Suppose not and given an instance of PML($\mathcal{L}$) if does not exists a model $M \in \mathcal{L}$ such that the instance $I = (\varepsilon, k)$ is not a YES instance there exists a model $M \in \mathcal{L}$ such that $f(I)$ is a YES instance for RAML. The model $M$ satisfies at least $k$ examples in $\varepsilon'$ which is identical to $\varepsilon$ (excluding the added agent identifiers). Thus, $M$ satisfies at least $k$ examples in $\varepsilon$ meaning that $I$ must be a YES instance of PML($\mathcal{L}$). A contradiction.

Since we know that PML($\mathcal{L}$) is NP-hard for $\mathcal{L}$ = PF, PT, or RPT (Theorems 6 and 7). We know that RAML($\mathcal{L}$) is NP-hard for $\mathcal{L}$ = PF, PT, or RPT. □

These complexity results mean we must still rely on our approximation techniques when learning preferences in a multi-agent context. One benefit of using simulated annealing for learning approximations is that modifying the algorithm to learn based on a new type of criterion simply means changing the evaluation function. By simply swapping the previous evaluation for a maximin score our algorithm will now attempt to optimize for the maximin criterion. Adapting our LPM baseline to the new multi-agent context is slightly more complex. While the greedy algorithm can remain the same for the utilitarian criterion it must be changed for the maximin criterion. For the maximin setting we decided to change the way that the algorithm selects attributes. In this new maximin greedy LPM learning algorithm the next element in the importance order is chosen based on the attribute which minimizes the incorrectly decided examples for the agent with the most incorrectly decided examples. Due to the intuitive appeal of this heuristic we select it as our baseline for our maximin learning experiments. That being said, there are no theoretical results to justify this choice.

Our experiments deal exclusively with a homogeneous set of agents, that is agents whose models are from the same language. In these settings we generate a random example set the same as we did before, but mark each example with the identifier of the agent that generated it. In our specific setting each group of agents consists of 5 random models of a given language. We applied the same cross validation strategy as in the single-agent setting, except we performed the folds on an agent by agent basis, thus each agent has a random $1/5^{\text{th}}$ of their examples removed from the training set rather than randomly removing $1/5^{\text{th}}$ of examples from the example set as a whole. Average training accuracy for the utilitarian setting is given in Table 5.4.

Table 5.4: Utilitarian accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.67 | 0.45 | 0.63 | 0.69 | 0.49 |
| 1,1,3-RPF | 0.46 | 0.56 | 0.59 | 0.51 | 0.51 |
| 2,2,7-RPF | 0.46 | 0.49 | 0.58 | 0.51 | 0.46 |
| 3,3,2,2,7-PT | 0.35 | 0.24 | 0.34 | 0.62 | 0.52 |
| 1,3,1,1,3-PT | 0.36 | 0.29 | 0.39 | 0.60 | 0.56 |
| 0-CP-net | 0.17 | 0.10 | 0.15 | 0.85 | 0.70 |
| 7-CP-net | 0.51 | 0.37 | 0.51 | 0.60 | 0.48 |

There is a reduction in training accuracy between the single-agent and multi-agent settings. This is most likely due to the introduction of contradictory examples, as agents may order alternatives differently. Unsurprisingly, we see that for the utilitarian criterion the general distribution of learning performance remains the same as in the single-agent setting. That is to say the same languages which performed well in the single-agent setting performed well in the utilitarian multi-agent setting. We see this trend continued in the average validation accuracy shown in Table 5.5.

Table 5.5: Utilitarian validation accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.63 | 0.42 | 0.51 | 0.58 | 0.43 |
| 1,1,3-RPF | 0.42 | 0.53 | 0.51 | 0.41 | 0.49 |
| 2,2,7-RPF | 0.43 | 0.46 | 0.48 | 0.40 | 0.42 |
| 3,3,2,2,7-PT | 0.31 | 0.21 | 0.26 | 0.46 | 0.44 |
| 1,3,1,1,3-PT | 0.33 | 0.27 | 0.31 | 0.44 | 0.50 |
| 0-CP-net | 0.15 | 0.08 | 0.10 | 0.73 | 0.66 |
| 7-CP-net | 0.47 | 0.34 | 0.42 | 0.44 | 0.39 |

The primary change observed between the single and multi-agent settings is the reduction in, and uniformity of, the variance wrt validation accuracy. As we see in Table 5.6 the standard deviation for validation accuracy is nearly uniform and much lower than what we saw in the single-agent setting, see Table 5.3. While some correlation with the generating model's language remains, any learned model has the same variance, wrt accuracy, on unseen examples. This means the likely outcome of learning joint preferences using the methods above will provide similar accuracy to what we see in Table 5.5.

Table 5.6: Utilitarian validation accuracy standard deviation when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.06 | 0.06 | 0.07 | 0.07 | 0.06 |
| 1,1,3-RPF | 0.06 | 0.07 | 0.07 | 0.07 | 0.07 |
| 2,2,7-RPF | 0.06 | 0.06 | 0.07 | 0.07 | 0.06 |
| 3,3,2,2,7-PT | 0.06 | 0.05 | 0.05 | 0.05 | 0.05 |
| 1,3,1,1,3-PT | 0.07 | 0.05 | 0.06 | 0.06 | 0.06 |
| 0-CP-net | 0.04 | 0.03 | 0.03 | 0.05 | 0.05 |
| 7-CP-net | 0.05 | 0.07 | 0.06 | 0.06 | 0.07 |

The other multi-agent setting we consider is evaluating solutions under the maximin criterion. The average training maximin scores produced by our learning algorithms are shown in Table 5.7.

Table 5.7: Maximin training accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.51 | 0.36 | 0.53 | 0.59 | 0.47 |
| 1,1,3-RPF | 0.33 | 0.39 | 0.48 | 0.41 | 0.45 |
| 2,2,7-RPF | 0.35 | 0.39 | 0.49 | 0.42 | 0.43 |
| 3,3,2,2,7-PT | 0.22 | 0.19 | 0.28 | 0.56 | 0.47 |
| 1,3,1,1,3-PT | 0.18 | 0.22 | 0.31 | 0.50 | 0.48 |
| 0-CP-net | 0.11 | 0.03 | 0.09 | 0.80 | 0.68 |
| 7-CP-net | 0.36 | 0.23 | 0.38 | 0.50 | 0.43 |

As we expect the accuracy of the models we learn under the maximin criterion is worse than in the utilitarian setting, however the reported scores are the proportion of examples satisfied for the least satisfied agent, rather than the average satisfaction per agent. Overall, at least $1/5^{\text{th}}$ of examples are satisfied for each agent in each setting, excluding ones with poor matches between the agents generating examples and the learned model (learning LPMs from CP-nets, for example). Thus, the maximin consensus models we generate are proportional, on average. Here we use proportional to mean that each agent has $1/n^{\text{th}}$ of their examples satisfied, where $n$ is the number of agents. This is a straightforward mapping of the concept of proportionality from the domain of division of goods, another social choice domain. For our specific experiments, a proportional aggregation is one where each agent gets at least $1/5^{\text{th}}$ of their examples satisfied. Of course, these results are for training accuracy. Table 5.8 shows how our maximin consensus models perform on unseen examples.

As we can see our naive proportionality holds even when considering unseen examples. Thus, our results provide evidence that using simulated annealing with a maximin criterion evaluation function is a viable method of aggregating preferences from a group of agents. However, if there is too much variance in the accuracy of the learned models it would be unlikely for proportionality to hold in practice. We see similar variance results as in the utilitarian setting, see Table 5.9.

Table 5.8: Maximin validation accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.42 | 0.24 | 0.35 | 0.51 | 0.43 |
| 1,1,3-RPF | 0.24 | 0.26 | 0.30 | 0.35 | 0.41 |
| 2,2,7-RPF | 0.27 | 0.28 | 0.30 | 0.35 | 0.41 |
| 3,3,2,2,7-PT | 0.16 | 0.10 | 0.13 | 0.31 | 0.29 |
| 1,3,1,1,3-PT | 0.13 | 0.11 | 0.15 | 0.27 | 0.29 |
| 0-CP-net | 0.06 | 0.01 | 0.02 | 0.72 | 0.65 |
| 7-CP-net | 0.29 | 0.17 | 0.25 | 0.39 | 0.36 |

Table 5.9: Maximin validation accuracy standard deviation when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | 0.07 | 0.07 | 0.09 | 0.06 | 0.07 |
| 1,1,3-RPF | 0.07 | 0.09 | 0.08 | 0.06 | 0.08 |
| 2,2,7-RPF | 0.06 | 0.07 | 0.09 | 0.07 | 0.06 |
| 3,3,2,2,7-PT | 0.07 | 0.06 | 0.06 | 0.07 | 0.08 |
| 1,3,1,1,3-PT | 0.07 | 0.06 | 0.06 | 0.07 | 0.08 |
| 0-CP-net | 0.04 | 0.02 | 0.03 | 0.05 | 0.05 |
| 7-CP-net | 0.07 | 0.07 | 0.08 | 0.06 | 0.06 |

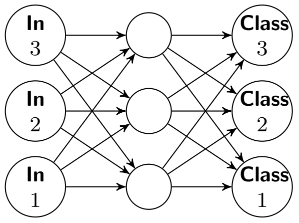
These results show simulated annealing works as a practical manner of learning RPFs and PTs in both single and multi-agent settings. For most settings, we found that learning an RPF or PT, on average, outperformed learning an LPM by a greedy algorithm, despite the algorithm's theoretical guarantees. As we expected there is a stratification in training accuracy between RPFs and PTs, given that PTs are more expressive than RPFs. A particularly interesting point of comparison is that when incomparable alternatives were present in the dataset, with CP-nets and CLPMs, PTs managed to have similar accuracy as when there were no incomparable pairs. There is no real comparison between PTs and RPFs, or LPMs, when it comes to reproducing incomparable pairs, since neither RPF nor LPM models can have incomparable alternatives, but the ability of PTs to maintain consistent average accuracy across many different types of example sets is encouraging.

Our simulated annealing techniques showed an interesting robustness. While we expect the utilitarian multi-agent setting to work similarly to the single-agent setting it was unknown how well optimizing for maximin would perform. For example, consider a candidate solution and its neighbors. There is a good chance that any given neighbor may improve some agent's satisfaction, but in order to be considered a better model it has to improve the least satisfied agent's satisfaction, while not sacrificing any other agent's satisfaction too much. This thought experiment implies there are more plateaus and other local search hazards when using a maximin criterion. Thus, it is interesting that when using the maximin criterion simulated annealing continues to perform well. This robustness may very well extend to other criteria, which is an area for potential future work.

## 5.2 Qualitative Preference Neural Networks

*Artificial neural networks* (ANNs) have proven to be robust and useful tools for dealing with a wide variety of learning tasks. Considering the diverse set of tasks to which ANNs have proven themselves to be well suited one would expect that if ANNs could be applied to learning preferences that they would be capable of learning preorders from many different languages. Our ANN approach learns qualitative preferences using linear feed-forward neural networks.

Figure 5.1: A simple multiclass linear feed forward neural network structure.



We have chosen to use linear feed-forward networks due to their simplicity. ANNs of this type are often used for classification problems and here we phrase qualitative preference learning as such. An example of a simple multiclass linear feed-forward network is given in Figure 5.1. In our case, for a given combinatorial domain $\mathcal{C}(\mathcal{V}, \mathcal{D})$, the input layer has a number of vertices equal to twice the number of attributes, $2|\mathcal{V}|$. The output layer has six output classes. This structure allows us to concatenate two alternatives together for input with the output class representing the relation between the two alternatives. The output classes consist of $\succ, \succeq, \approx, \prec, \preceq$, and $\bowtie$.

Given an example $(\alpha, \beta, R)$ we input $\alpha\beta$ (the concatenation of the two alternatives) and train the network to return $R$ as the class label. Using ANNs in this manner has some drawbacks. For one, it is possible that if we input $\alpha\beta$ we get one relation and when we input the reverse, $\beta\alpha$, we do not get the inverse relation. For example if we input $\alpha\beta$ we might get the label $\succ$ meaning $\alpha \succ \beta$, but upon inputting $\beta\alpha$ we might get the label $\succ$ meaning $\beta \succ \alpha$. To avoid such circumstances we order the alternatives before they are used as input. Specifically, they are ordered using a lexicographic ordering where $v_1 \succ v_2 \succ \ldots \succ v_n$ and lower values are preferred to larger ones on each domain. While presorting the input in this fashion removes the above issue, it does mean that the trained ANNs may be biased towards representing orders which closely match the presorting.

Another problem present with the use of ANNs is that the preference ordering the ANN induces is not guaranteed to be closed under transitivity. In other words, the network may learn the following three relations: $\alpha \succ \beta$, $\beta \succ \gamma$, and $\gamma \succ \alpha$. If we assume transitivity then $\alpha \succ \alpha$, which is impossible if $\succ$ is a strict order derived from a preorder. While we were able to remedy the input ordering problem, we are not aware of any way to fix the transitivity problem. Thus the ANNs we train do not always induce a preorder relation. Unlike the formal languages we have discussed earlier, which induce preorders, ANNs can represent seemingly contradictory information. This could be advantageous from an accuracy standpoint since inconsistent information can be modelled, but given that preorders are how we have chosen to represent preferences ANN models can fall short of that goal.

We tested the effectiveness of using ANNs with the same approach we used for simulated annealing. The primary difference between the ANN models we consider is the number of hidden layers used. For our purposes we tested networks with $0, 1, 2,$ and 3 hidden layers, given that the number of hidden layers affects an ANN's approximation abilities [Hornik, 1991]. In all cases a hidden layer consists of 256 neurons which is the same as the number of alternatives in the combinatorial domain used for example generation. This number was chosen because it represented a reasonably large layer size which gives our models a better chance at higher learning accuracy. Training and construction was done using the Pytorch python module [Paszke et al., 2017]. Specifically, we used stochastic gradient descent with minibatch updates where each batch consisted of 10 examples. Each model was trained for 1000 epochs which was selected because it represented, in our tests, a point after which there was little further improvement in training accuracy.

Table 5.10: Neural network learning accuracy in single-agent settings.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 1.00 | 0.93 | 0.83 | 0.80 | 0.83 |
| 1,1,3-RPF | 0.64 | 0.89 | 0.74 | 0.70 | 0.78 |
| 2,2,7-RPF | 0.61 | 0.84 | 0.68 | 0.64 | 0.73 |
| 1,3,1,1,3-PT | 0.51 | 0.69 | 0.72 | 0.69 | 0.63 |
| 3,3,2,2,7-PT | 0.47 | 0.64 | 0.68 | 0.68 | 0.67 |
| 0-CP-net | 0.19 | 0.94 | 0.92 | 0.90 | 0.92 |
| 7-CP-net | 0.73 | 0.86 | 0.80 | 0.76 | 0.76 |
| 0-CLPM | 0.53 | 0.87 | 0.88 | 0.86 | 0.82 |
| 7-CLPM | 0.53 | 0.86 | 0.89 | 0.86 | 0.84 |
| 1,1-LP-tree | 0.92 | 0.90 | 0.89 | 0.88 | 0.85 |
| 3,3-LP-tree | 0.91 | 0.89 | 0.87 | 0.88 | 0.84 |

From a training perspective, ANNs produce very accurate results, see Table 5.10. It is interesting that the number of hidden layers does not seem to affect the accuracy of ANN models. This implies that simpler networks are just as capable of representing preferences as more complex ones.

Additionally, we find that ANNs without hidden layers have greater accuracy than those with three, which is the opposite of what we expected. There are several possible reasons for this, though we conjecture that preferences, which come from highly structured representations and have structured properties, provide a rather simple function to approximate, especially given that the input comes from a discrete, high-dimensional domain (combinatorial domain), rather than a real-valued one.

Table 5.11: Neural Network validation accuracy in single-agent settings.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 0.98 | 0.85 | 0.77 | 0.73 | 0.76 |
| 1,1,3-RPF | 0.62 | 0.80 | 0.66 | 0.64 | 0.71 |
| 2,2,7-RPF | 0.55 | 0.59 | 0.55 | 0.52 | 0.52 |
| 1,3,1,1,3-PT | 0.49 | 0.54 | 0.57 | 0.56 | 0.49 |
| 3,3,2,2,7-PT | 0.40 | 0.49 | 0.53 | 0.53 | 0.53 |
| 0-CP-net | 0.18 | 0.79 | 0.81 | 0.80 | 0.79 |
| 7-CP-net | 0.65 | 0.63 | 0.60 | 0.58 | 0.56 |
| 0-CLPM | 0.51 | 0.68 | 0.66 | 0.69 | 0.70 |
| 7-CLPM | 0.50 | 0.69 | 0.71 | 0.67 | 0.69 |
| 1,1-LP-tree | 0.84 | 0.82 | 0.78 | 0.77 | 0.74 |
| 3,3-LP-tree | 0.82 | 0.77 | 0.76 | 0.76 | 0.72 |

When applied to unseen examples, see Table 5.11, we continue to see good accuracy from ANNs. While accuracy is decreased, compared to training, it is still very high, with no settings having an average less than 0.50, meaning at least half of the unseen examples were satisfied on average. Note that compared to our previous simulated annealing approach these results do tend to have a steeper drop between training and validation. This implies that some overfitting is occuring, or at least, more than when learning RPFs and PTs.

Looking at the standard deviation among the validation accuracy, see Table 5.12, we can see a similar effect to what we saw with simulated annealing, i.e. the language

Table 5.12: Neural Network validation accuracy standard deviation in single-agent settings.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 0.04 | 0.19 | 0.19 | 0.20 | 0.19 |
| 1,1,3-RPF | 0.12 | 0.26 | 0.26 | 0.25 | 0.26 |
| 2,2,7-RPF | 0.13 | 0.16 | 0.17 | 0.17 | 0.17 |
| 1,3,1,1,3-PT | 0.16 | 0.18 | 0.16 | 0.18 | 0.18 |
| 3,3,2,2,7-PT | 0.14 | 0.13 | 0.13 | 0.15 | 0.13 |
| 0-CP-net | 0.09 | 0.09 | 0.09 | 0.09 | 0.09 |
| 7-CP-net | 0.12 | 0.16 | 0.15 | 0.16 | 0.16 |
| 0-CLPM | 0.22 | 0.16 | 0.16 | 0.18 | 0.19 |
| 7-CLPM | 0.24 | 0.16 | 0.16 | 0.16 | 0.16 |
| 1,1-LP-tree | 0.09 | 0.14 | 0.16 | 0.18 | 0.18 |
| 3,3-LP-tree | 0.09 | 0.15 | 0.15 | 0.16 | 0.17 |

of the model that generated the example set tends to dictate the variance of the validation accuracy. It is important to note that ANNs tend to produce a greater amount of variance than other approaches. For the single-agent setting this means that the case-by-case accuracy is less reliable than when using a model from a classical preference language.

Using ANNs has proven adequate in the single-agent setting, but how does it perform as a consensus model? One advantage ANNs have is the ability to hold contradictory information. As we mentioned above, this increased expressive power could lead to higher average accuracy than we observed when using simulated annealing, but comes at the price of not inducing a preorder.

Table 5.13: Utilitarian accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 0.67 | 0.76 | 0.79 | 0.79 | 0.79 |
| 1,1,3-RPF | 0.46 | 0.68 | 0.66 | 0.67 | 0.67 |
| 2,2,7-RPF | 0.46 | 0.66 | 0.67 | 0.67 | 0.66 |
| 0-CP-net | 0.17 | 0.89 | 0.90 | 0.89 | 0.90 |
| 7-CP-net | 0.51 | 0.70 | 0.70 | 0.70 | 0.69 |

Table 5.13 shows that there is an improvement, wrt to training accuracy, over the simulated annealing and greedy LPM strategies. Also, in training the trend of similar results for the different depths of network holds.

Table 5.14: Utilitarian validation accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 0.63 | 0.56 | 0.57 | 0.56 | 0.57 |
| 1,1,3-RPF | 0.42 | 0.44 | 0.42 | 0.42 | 0.42 |
| 2,2,7-RPF | 0.43 | 0.40 | 0.40 | 0.40 | 0.41 |
| 0-CP-net | 0.15 | 0.78 | 0.78 | 0.77 | 0.79 |
| 7-CP-net | 0.47 | 0.43 | 0.43 | 0.44 | 0.43 |

The accuracy drop going from training to validation, see Table 5.14, is larger for the multi-agent setting than in the single-agent setting. In some cases, LPMs are more accurate than ANNs. One possible explanation is that the ANNs are overfitting and failing to generalize. Another is that we are trying to approximate something which is not well modelled by linear feed-forward ANNs.

Table 5.15: Utilitarian validation accuracy standard deviation when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 0.06 | 0.06 | 0.06 | 0.06 | 0.07 |
| 1,1,3-RPF | 0.06 | 0.08 | 0.09 | 0.07 | 0.07 |
| 2,2,7-RPF | 0.06 | 0.06 | 0.05 | 0.06 | 0.07 |
| 0-CP-net | 0.04 | 0.05 | 0.05 | 0.06 | 0.05 |
| 7-CP-net | 0.05 | 0.07 | 0.05 | 0.06 | 0.06 |

As for the variance associated with the validation accuracy of the learned models, we see the same trend as before: low, uniform variance in all settings. Given this data it appears that ANNs produce a sufficiently accurate picture of a consensus preference. That being said, there are improvements that could be made, for example different hyperparameters and/or network structures could render a model which is better at generalizing and so retains greater accuracy on unseen examples.

ANNs produce high accuracy results for the utilitarian aggregation, but what of the maximin aggregation? Unlike the process of converting the greedy algorithm or simulated annealing to handle maximin aggregation there is no easy way to train an ANN according to the maximin criterion. We are unaware of any work in this direction and its development is beyond the scope of this work. With that in mind, we computed the maximin score of each network trained under the utilitarian criterion. While this is not a true test of maximin performance for ANNs it does give some insight into how egalitarian the learned consensus models are.

Table 5.16: Maximin training accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 0.51 | 0.67 | 0.71 | 0.69 | 0.71 |
| 1,1,3-RPF | 0.33 | 0.60 | 0.58 | 0.61 | 0.59 |
| 2,2,7-RPF | 0.35 | 0.59 | 0.59 | 0.60 | 0.57 |
| 0-CP-net | 0.11 | 0.85 | 0.87 | 0.85 | 0.86 |
| 7-CP-net | 0.36 | 0.61 | 0.61 | 0.62 | 0.60 |

Table 5.16 shows the average maximin score for the ANNs. Note that while there is some loss in accuracy, the numbers still indicate that ANNs do very well in terms of training accuracy, even for the least satisfied agent. We see a slightly more pronounced difference between ANNs of different depths in this case. It appears that the addition of more layers causes a more egalitarian consensus model, although this effect is rather small.

Table 5.17: Maximin validation accuracy when learning joint preferences.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 0 Hidden | 1 Hidden | 2 Hidden | 3 Hidden |
| LPM | 0.42 | 0.40 | 0.42 | 0.41 | 0.41 |
| 1,1,3-RPF | 0.24 | 0.28 | 0.28 | 0.28 | 0.28 |
| 2,2,7-RPF | 0.27 | 0.25 | 0.26 | 0.25 | 0.25 |
| 0-CP-net | 0.06 | 0.66 | 0.66 | 0.66 | 0.67 |
| 7-CP-net | 0.29 | 0.27 | 0.27 | 0.29 | 0.28 |

Moving to the validation set we see that the results are very poor. Often it is better to go with a greedily learned LPM rather than an ANN. In settings where LPMs are less accurate, we note that learning PTs using simulated annealing is more accurate. The loss of accuracy between training and validation is more pronounced when we apply the maximin criterion, rising to an almost 0.30 difference in some settings. While not trained in a maximin context, this indicates that ANNs are a poor choice for maximin preference aggregation. This may change if a maximin training protocol were to be developed.

## 5.3 Limitations and Further Work

Overall, both simulated annealing and ANNs provide good approaches for learning qualitative preferences. We find that both PTs and ANNs represent sufficiently “universal” preference learners, that is they are capable of learning many different types of preferences. Of course, there were situations where both the LPMs and RPFs had greater accuracy, but without the ability to have pairs of incomparable alternatives both LPMs and RPFs have a known weak point that makes them less universal languages for preference learning. The data generated by the experiments above provides a good baseline for further work.

One direction for further work is to increase the number of languages used to generate example sets. Another is to increase the number of examples used for training considering the tendency for ANN to require large datasets to be effective and the datasets we constructed were small by comparison. The use of larger sets may result in better performance and is an area for future study.

A limitation of this work is that we do not determine the degree to which ANNs induce “flawed” preference orderings. Tackling this problem focuses on the application of transitivity to the local (pairwise) comparisons that the ANN learns. As was briefly discussed above, an ANN is not guaranteed to build a relation that is closed under transitivity. Analyzing how “flawed” an ANNs preference ordering is requires defining how close an ANN’s relation is to a preorder. One such measure is counting how many alternatives are strictly preferred to themselves. Thus the measure indicates how much the network violates transitivity. The proposed metric is one of many we might consider and this is a place for future work.

These flaws also indicate why it may be advantageous to learn a PT model over an ANN model even if we might see a decrease in accuracy when using a PT model. This argument can be further extended to consider the advantages of other representations. For example, determining optimality for a PT is a coNP problem [Brewka et al., 2003], for LPMs the problem of optimality is in P. This means that selecting the type of preference representation to learn may be dependent on more than the accuracy of the final model.

We reiterate that both of these learning techniques produce approximations of the optimal models and that both problems are computationally difficult. For simulated annealing we prove this ourselves. For ANNs, the learning task has been shown to be NP-complete in other settings [Bartlett and Ben-David, 1999]. We expect these results transfer to our setting, given we see no simplifications being attached to our problems. Since we are dealing with approximations and more importantly metaheuristics, there are many different hyperparameters. Changes to these hyperparameters may increase or decrease accuracy and we leave it to better optimizers to find the best settings for learning preferences.

A final piece of future work in this domain is finding a way to perform maximin learning on an ANN. While we find it most applicable for learning consensus preferences there is no doubt other applications could benefit from such a learning procedure.

# Chapter 6 Classification of Example Sets and Portfolio Learning

There are two primary approaches to learning preferences without the drawbacks of *a priori* committing to a language to learn a model from: (1) select a “universal” language and (2) automate selecting the appropriate language on a case-by-case basis. The simpler option is selecting a single “universal” language. For our purposes universal means a language whose models can replicate a wide variety of preference orders well. Given a universal language, which is efficiently learnable, the focus for learning preferences would be on improving its learning algorithms.

To identify potential universal languages we examined both our subsumption analysis and our previous learning work. While our data is limited to the subset of languages we selected, we find that according to both subsumption and our experiments PTs appear to be a sufficiently universal preference representation language. From a purely experimental perspective we also find that artificial neural networks (ANNs) can be trained to offer good accuracy over example sets from a broad range of preference models. While these languages appear to be universal, in terms of the preference orders they can represent, it is important that we analyze the data to see what role language selection plays in learning preferences and whether or not the case-by-case method of language selection has any merit. Broadly speaking this defines the *language selection problem* which we formally define below.

**Problem 6** (Language Selection)**.** *Given an integer $k$, a set of preference representation languages $\boldsymbol{L} = \{\mathcal{L}_1, \dots, \mathcal{L}_n\}$ and an example set $\varepsilon$ over a domain $\Omega$ determine which language(s) $\mathcal{L} \in \boldsymbol{L}$ contains a model $M \in \mathcal{L}$ such that $M$ satisfies at least $k$ examples in $\varepsilon$.*

## 6.1 Correlation Between Generating Model and Learned Model

Let us consider the learning experiments done previously. For the single-agent setting we made some simplifying assumptions. First, all our examples come from agents whose preferences are defined by classical preference models. Second, the example sets used for learning are generated from preorders (meaning they are transitive). Third, these example sets contain examples that were generated at random and are relatively small (compared to the space of all possible examples.) Such assumptions may not hold in real world data.

Table 6.1: Validation accuracy in single-agent settings.

| | Learned Model Type | | | | |
|---|---|---|---|---|---|
| Agent Model | LPM | 1,1,3-RPF | 2,2,7-RPF | 3,3,2,2,7-PT | 1,3,1,1,3-PT |
| LPM | **0.98** | 0.60 | 0.75 | 0.80 | 0.61 |
| 1,1,3-RPF | 0.62 | **1.00** | 0.98 | 0.62 | 0.84 |
| 2,2,7-RPF | 0.55 | 0.68 | **0.94** | 0.54 | 0.58 |
| 3,3,2,2,7-PT | 0.40 | 0.27 | 0.32 | **0.59** | 0.57 |
| 1,3,1,1,3-PT | 0.49 | 0.40 | 0.48 | 0.74 | **0.98** |

One pattern exists throughout our data: we obtain the highest accuracy if the language of the learned model is the same as the language of the model that produced the examples. At least, for the three languages we investigated: LPM, RPF, and PT. This holds true for both training and validation accuracy, see Table 6.1. Therefore, given an example set and the language of the model used to create it we can expect greater learning accuracy by selecting the matching language.

To realize such an approach we need the ability to classify example sets based on their language of origin. We could then select the appropriate learning algorithm for the example set from a known list. This approach is a version of the algorithm selection problem [Rice, 1976]. The algorithm selection problem concerns selecting the best algorithm to solve a particular instance of a problem, from a set of such algorithms, where different algorithms perform better on different instances of the problem.

A popular solution to the algorithm selection problem is the use of portfolios. A portfolio solution involves collecting a number of algorithms and a method of predicting the runtime of each algorithm for a particular problem instance [Leyton-Brown et al., 2003]. By predicting the runtime for a particular instance the system can choose the fastest algorithm to use for that instance. This approach has been shown to be useful in SAT solving [Xu et al., 2008] and answer set programming [Gebser et al., 2011], which are both NP-complete, or harder, problems. We will discuss a potential preference language selection system later for a portfolio preference learning system.

Another data trend in our learning experiments expands on the earlier observation where the learned model is of the same langauge as the modeled that generated the example set. Namely, we observe greater accuracy for languages that are closer to each other, according to the subsumption relation. For example, the PT language subsumes the RPF language and the data shows that PTs have higher accuracy when learning example sets generated by RPFs than LPMs. Our data supports the idea that the closer two languages are the more accurate the learned model.

We see that both PTs and ANNs perform well in the general learning of preferences and attribute it to their ability to induce many different preference orderings. When selecting a language to model an agent's preference considerations other than the accuracy of the learned model exist. For example, consider two models, one an RPF and the other an ANN, which have similar training and predictive accuracy. We argue that the RPF provides a more useful model than the ANN, even with worse accuracy. This argument comes from the ability to extract preference information (other than individual comparrisons) from an RPF. For example one can extract a formula for maximal alternatives given an RPF, such a task is impossible for an ANN.

Based on the above results, the problem of language selection can be considered one of example set classification, i.e. if we can classify an example set by the language of the model that generated it we may be able to increase our learning accuracy by selecting that language. While there are many approaches to such a task and we focus on two: a linear feed forward ANN and a graph convolutional neural network (GCNN). We selected these two approaches given that they represent two important classes of input. The first class of input, represented by the ANN, is that which transforms an example set into a series of features. Thus, our selection of features impacts how well the system can classify example sets. The second class of input, represented by the GCNN, is simply using the entire example set without any (or minimal) preprocessing. Thus, we make no decision over which information might be important before applying our classifier.

We divide our experiments into three categories, based on the perceived difficulty of the task. The first (referred to as the "simple" dataset) is classifying example sets between two very different languages, specifically LPMs and CP-nets, which we expect to be simple since the preference orders induced by LPMs and CP-nets are disjoint and the languages are incomparable according to subsumption. The second (referred to as the "similar" dataset) is classifying example sets between two similar languages, specifically CLPMs and CP-nets, which we expect to be much harder given that CLPMs are capable of approximating CP-nets. The third (referred to as the "full" dataset) is classifying example sets between seven different languages, which aligns more closely to what we might expect for a fully realized portfolio-based preference learning system. Taken together these three tests allow us to reason about the overall efficacy of using a particular approach for classifying example sets.

## 6.2 Linear Feed Forward Neural Networks

Our first concern for building any machine learning system is data representation. Any representation used should present enough information to distinguish instances from one another, but also ensure that the patterns learned are due to the intrinsic properties of an instance rather than random occurrence. Therefore, we must consider which features of an example set are pertinent to our classification task. These features should reflect the language of the model which generated the example set since that is what we are trying to learn. For our experiments we looked at two such representations, called feature vectors. The first of these feature vectors is independent of the domain of alternatives while the second extends the first by adding domain specific information.

Our basic feature vector consists of the proportion of examples in a given example set associated with a particular relation from among $\succ, \succeq, \approx, \preceq, \prec,$ and $\bowtie$. This means our feature vector consists of six floating point values between 0 and 1. We chose these particular features because (1) we can determine which relations are needed to generate the example set and (2) the models of a language tend to produce similar proportions of specific types of examples. Speaking to the second point, we noticed that most CP-nets produce $20\% - 40\%$ incomparable pairs in their examples sets while PTs produce $40\% - 50\%$ (for randomly generated models). Thus, relations are a natural starting place to build a feature vector.

We trained linear feed forward neural networks which varied in the number of hidden layers, between 0 and 3, while keeping the number of neurons in the hidden layers constant at 256, this number was chosen as a starting point since it is the same as the number of alternatives in the domains we tested against (each domain was a combinatorial domain with eight binary attributes). Since larger hidden layers tend to increase the tendency for an ANN to overfit this choice gave us room to downsize if we saw such effects. The examples used to train our ANNs consisted of a pair containing a language label and a feature vector. Each language classification example consists of a pairwise preference comparison example set and the language of the model that created it. To build a language classification example we first choose a language, then construct a random model of that langauge. Using the model we build a pairwise prefernce comparison example set of 100 examples. The pairwise preference comparison example set is then converted into a feature vector and labeled with the language that we selected in the beginning. We created 400 langauge classification examples for each of the langauges we consider.

As mentioned above, we deal with three main types of datasets labelled simple, similar, and full. The simple dataset consists of language classification examples from either LPM or acyclic CP-net models. These languages are disjoint and therefore should be easy to distinguish. They are also relatively easy for a human to correctly classify given that if any incomparable alternatives exist in an example set we know that that feature vector must represent a CP-net since LPMs cannot have incomparable alternatives.

Datasets of the second type ("similar") consist of examples generated by CLPM or CP-net models. Since CLPMs can approximate (and in some cases are equal to) CP-net models. This provides a setting where it may be difficult to tell which language contains the model that generated a particular example set. The similar dataset helps us to understand where potential mistakes might be made by the trained ANNs. Finally, we have the full dataset. The full dataset consists of examples generated by 7 different languages: LPMs, CP-nets, Weighted Averages, Penalty Logic, RPFs, LP-trees, and CLPMs. The full dataset better represents the kind of scenario this technique might run into in the real world.

Table 6.2: Average training accuracy for basic feature vector.

| | Model Type | | | |
|---|---|---|---|---|
| Language set | 0 Hidden Layers | 1 Hidden Layer | 2 Hidden Layers | 3 Hidden Layers |
| Simple | 1.00 | 1.00 | 1.00 | 1.00 |
| Similar | 0.66 | 0.67 | 0.67 | 0.68 |
| Full | 0.71 | 0.74 | 0.75 | 0.74 |

As expected we see high accuracy for the simple datasets, see Table 6.2. There is a decrease in accuracy for the similar dataset over the simple and full datasets. While the accuracy is at 0.66 it is important to note that random chance gives an accuracy of 0.5 since there are two options and a balanced number of examples. This means that the learned classifier correctly identifies the language associated with a particular feature vector with a higher likelihood than guessing. While we see an increase in accuracy for the full dataset we observed that most of the mistakes made in the full datasets come from the confusion of similar languages, specifically CLPMs and CP-nets. While training scores can be high we want to find models which generalize to unseen examples. To test this we implemented five-fold cross validation, the same type we used for testing our preference learning algorithms. The data is shown in Table 6.3.

Table 6.3: Average validation accuracy for basic feature vector.

| | Model Type | | | |
|---|---|---|---|---|
| Language set | 0 Hidden Layers | 1 Hidden Layer | 2 Hidden Layers | 3 Hidden Layers |
| Simple | 1.00 | 1.00 | 1.00 | 1.00 |
| Similar | 0.66 | 0.65 | 0.65 | 0.66 |
| Full | 0.71 | 0.73 | 0.73 | 0.73 |

These results indicate that the learned models work similarly on both seen and unseen examples. Despite good performance on seen and unseen examples, there is room for improvement in accuracy. To explore the possibility of an improvement we expanded the basic feature vector by adding a feature for each attribute-value pair. This feature is the proportion of examples where that attribute-value pair is in the dominant alternative. For example if alternatives with value 2 for attribute 1 show up in 10 examples with the relation being $\succ$ or $\prec$ and in 8 of those examples the alternative with the value 2 for attribute 1 is in the preferred alternative then the feature for the $1, 2$ attribute-value pair is set to 0.8. We generate these features for all attribute value pairs in the domain. Since in our experiments we consider a binary combinatorial domain over eight attributes, this adds 16 additional features to our feature vector.

The addition of these features was motivated by LPMs where certain attributes are more likely to determine dominance than others. For example if the most important attribute of an LPM is attribute 4 and the attribute domain preference order is $1 \succ 0$ then alternatives with the value 1 for attribute 4 are more likely to be in the dominant alternative in an example. Such trends might also exist in small example sets. When these additional features were added to our input feature vector it produced the following results, see Table 6.4.

Table 6.4: Average training accuracy for expanded feature vector.

| | Model Type | | | |
|---|---|---|---|---|
| Language set | 0 Hidden Layers | 1 Hidden Layer | 2 Hidden Layers | 3 Hidden Layers |
| Simple | 1.00 | 1.00 | 1.00 | 1.00 |
| Similar | 0.66 | 0.88 | 0.99 | 0.99 |
| Full | 0.74 | 0.84 | 0.93 | 0.91 |

These results indicate the additional data does help with classification. As we can see in the 2 and 3 hidden layer settings this additional information even helps to remove the ambiguity when comparing similar languages. While we dont quite see the same extreme increase in the full dataset as in the similar data, the increase is promising. This story changes when we look at how these models perform on unseen examples, see Table 6.5.

Table 6.5: Average validation accuracy for expanded feature vector.

| | Model Type | | | |
|---|---|---|---|---|
| Language set | 0 Hidden Layers | 1 Hidden Layer | 2 Hidden Layers | 3 Hidden Layers |
| Simple | 1.00 | 1.00 | 1.00 | 1.00 |
| Similar | 0.65 | 0.65 | 0.63 | 0.62 |
| Full | 0.72 | 0.74 | 0.73 | 0.73 |

Predicting unseen examples using the expanded feature vector shows a tendency to overfit. We see that the learned models are actually slightly worse, on average, than their counterparts learned using the smaller feature vectors. This effect has two possible causes. The first is that the attribute-value pair proportions are relatively randomly distributed and so the networks are simply picking up on very specific, non-generalizing cues. The second is that the networks are being overtrained, or more generally our choice of hyperparameters is not optimal. Since the validation accuracy of both feature vectors is similar it is reasonable to say that the additional information has no practical benefit and so creates learning cues which do not generalize. There is another downside to using our extended feature vector: it is domain dependent, that is, it's size depends on the combinatorial domain we are dealing with. A domain dependent feature vector means that if we change the domain of interest we need to retrain our model from scratch as where it may be possible to use a domain independent classifier (such as one trained using our smaller feature vector) trained on a smaller domain for use in a larger domain.

## 6.3 Graph Convolutional Neural Networks

Although linear feedforward ANNs proved useful there is still room for improvement. In the ANN literature there are many ANN structures which make use of image convolutions. Image convolutions are a processing pass over image data which transforms the input (typically a block of several pixels) into some useful value, or set of values. The use of convolutions in ANNs has allowed for improvements in many image based tasks, such as image recognition [LeCun et al., 1998]. These techniques have been modified to work with graph structured data [Scarselli et al., 2008].

Given that any binary relation can be represented using a graph we can convert example sets into graphs allowing us to apply graph convolutions as part of our classification process. We do so by learning graph convolutional neural networks (GCNNs). Before we condensed example sets into vectors of features; but with GCNNs the entire graph is used as input. This means our classifier is trained with raw preference data rather than preprocessed feature vectors.

Our GCNN models have several convolutional layers followed by an equal number of normal linear layers (the same type of layers used in our previous experiments) followed by an average pooling layer, which converts the high dimensional graph data down to one dimension by averaging. Our models are specified by the number of convolutions used. This means that a 2 layer GCNN model consists of 2 graph convolution layers and 2 linear layers for a total of 4 layers. Like our previous experiments we tested 1, 2, and 3 layer GCNNs (we did not consider 0 layer GCNNs as they are not well defined.) For our convolutions we used those detailed by Kipf and Welling [2017], since they deal with graph classification, and use the implementation by Fey and Lenssen [2019] in the Pytorch Geometric Python module.

We convert a pairwise preference comparison example set into a graph $G = (V, E)$ by first setting $V$ equal to the set of alternatives in the example set. We then add in edges such that $(u, v) \in E$ if $u \succeq v$. Thus, each $\succ$ or $\prec$ example adds one arc, $\approx$ examples adds two arcs, and $\bowtie$ add none.

When initially testing our system we noticed near zero accuracy from our classifier (for our full dataset), even after 2000 training epochs. We realized that our randomly generated example sets consist of 100 random pairs of alternatives. Considering there are 256 total alternatives in our domain, the graphs the example sets specify were mostly large sets of disjoint 2-vertex graphs since there is litte chance that two examples share a common alternative. We concluded that this (lack of) structure does not provide much information to our classifier. With this in mind we modified the way we generate pairwise comparison example sets to (1) provide more graph information and (2) maintain a similar example set size to our previous method (i.e. 100 pairwise comparisons).

Our pairwise comparison example set generation for use with a GCNN classifier starts the same as our normal example set generation. We select a language and construct a random model from that langauge. Instead of selecting 100 random pairs of alternatives we select 14 individual alternatives. We then generate an example for each unique pair of those 14 alternatives, avoiding comparing an alternative with itself. This gives us 91 pairwise comparison examples in our example set. The example set is then converted into a graph as detailed above. By structuring our example sets this way we build a graph which is closed under transitivity and contains more graph information since more vertices interact with one another.

Our GCNN models were then tested the same way as our feature vector ANNs. That is to say we applied five-fold cross validation and the same datasets (simple, similar, and full) as above. The result of these experiments is shown in Table 6.6.

Table 6.6: Average training accuracy for GCNN models.

| | Model Type | | |
|---|---|---|---|
| Language set | 1 Layer | 2 Layers | 3 Layers |
| Simple | 0.56 | 0.57 | 0.67 |
| Similar | 0.59 | 0.59 | 0.59 |
| Full | 0.51 | 0.52 | 0.47 |

These results show that learning an example set classifier using GCNNs is not a promising approach. Possible alternatives might include using larger pairwise comparison example sets, an increase in layer size and/or layer composition, or further tuning of GCNN hyperparameters.

## 6.4 Portfolio-based Preference Learning

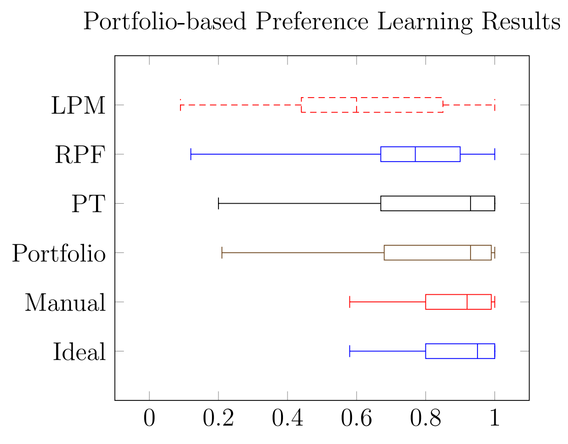
Combining our feature vector classifier approach with a small collection of preference learning algorithms we constructed a basic portfolio-based preference learning system. This system takes as input a set of examples, converts it into a feature vector, runs that vector through a classifier which selects a preference representation language, and then runs a learning algorithm for the selected language which learns a model of the selected language based on the example set. Previously, we have conjectured about the usefulness of such a system, here we test this idea (in a limited setting).

We constructed this system to select from among the languages of LPM, RPF, or PT. The central component of our learning system is the classifier which tells us which language to select. Given the success of our feature vector approach, where an example set is reduced to the proportion of types of examples, we have chosen to use it as the basis for our classifier.

The classifier's training set was constructed from 25 language classification examples per training language with the following training languages: CP-net, CLPM, Weighted Average, LP-tree, Penalty Logic, RPF, PT, and LPM. We constructed our classification examples by first generating a random model of a langauge. We then select 100 random pairs of alternatives (in this case our combinatorial domain consists of eight binary attributes) and use the model to generate 100 pairwise comparison examples. This set of pairwise comparisons is then converted into a feature vector containing the proportion of the various relations that are present in the example set. Instead of labeling the feature vector with the language of the model that created it, we apply each of our learning algorithms to see which best learns the example set. Our learning languages are $2, 2, 7$-RPF, $3, 3, 2, 2, 7$-PT, and LPM. The length settings for RPFs and PTs were chosen because they represent the most complex settings we use in this work, but also because larger PTs and RPFs are capable of closely approximating less complex PTs and RPFs as our learning data shows. Each feature vector is labeled with the language whose learning algorithm best learned its associated example set. By training the classifier in this manner it is trained to determine which of our learning algorithms works best for a given example set.

Overall, our learning system takes as input an example set from some agent; converts those examples into a feature vector; classifies that feature vector as being best learned by either a $2, 2, 7$-RPF, $3, 3, 2, 2, 7$-PT, or LPM; and learns a model from the language indicated by the classifier using the given example set. In order to evaluate this learning system we constructed an additional 25 pairwise comparison example sets from each of the training languages and used our system to learn a preference model for each example set. Since we know that for learned RPF, PTs, and LPMs the difference between performance on seen and unseen examples is very similar we only report the training accuracy and did not perform cross validation. We also evaluated five other learning schemes. Three of these consisted of always selecting the same language (no matter the example set.) The fourth scheme ("ideal") consisted of learning a $2, 2, 7$-RPF, $3, 3, 2, 2, 7$-PT, and LPM and then selecting the highest performing model. The fifth scheme ("manual") consisted of a hard coded system which learned a $3, 3, 2, 2, 7$-PT if the example set had incomparable alternatives; a $2, 2, 7$-RPF if the example set had no incomparable alternatives, but had equally preferred alternatives; and finally an LPM in all other cases. The distribution of training accuracy over the pairwise comparison example sets is shown in Figure 6.1

Figure 6.1: Distribution of training accuracy across 6 learning schemes.



These results show that our hard coded language selector outperformed our machine learning approach and closely resembled the ideal scheme, although it had a lower median than both the portfolio and ideal schemes. While this approach worked well for this particular system it is not a viable method, in general. For example, consider a system which could learn many languages that could produce incomparable alternatives, then it would be difficult to decide which language to use when incomparable alternatives exist in the example set. In such a case we may need a system which is capable of parsing the nuance between example sets. In other words, our manual language selection scheme worked well because there was a clear delineation between the languages we could learn that was represented in the example sets.

That being said, when we applied our portfolio system with a classifying ANN we see performance which is close to the scheme where we learn a PT in all cases. While we do not see a major performance benefit over simply always learning a PT, there is another dimension to our portfolio system. The PT scheme will always learn a $3, 3, 2, 2, 7$-PT, even if an equally effective LPM could be learned. In this case it would be easier to extract information such as optimal and pessimal alternatives from the LPM than from the PT. Additionally, it takes less time to learn an LPM than a $3, 3, 2, 2, 7$-PT. This means that our portfolio system has some advantages over the PT learning scheme despite similar accuracy performance.

While our portfolio-based prefence learning system did not outperform the ideal and its language selection (in this particular instance) can be effectively hard coded, it does show promise as a preference learning techinque. We find that learning an example set classifier needs additional work and perhaps could achieve performance rivaling our hard coded language selection system.

## 6.5 Limitations

The process of learning which language contains the model that generated a particular example set has inherent limitations. The primary limitation is that it relies on a large amount of data. In all cases we must build example sets and each example set may be large and/or complex. These example sets then need to be reduced down into a form which can be analyzed. For each example set we generate we only get one training example so the process must be repeated many times. This makes training set generation for example set classification a time consuming process.

Another limitation, or more accurately challenge, of building an example set classifier is that we must select which languages to include. Any non-exhaustive selection of languages implies the ability for the misidentification of which language an example set comes from. For example if we learn the difference between LPMs and RPFs, but give the classifier an example set from a CP-net then it may select the wrong language to use. Unfortunately, there is no fool proof way to avoid this problem, but it is something we must be aware of, like how language similarity may cause inaccuracies in classification.

While our portfolio learner might be improved by increasing the number of learning algorithms we use, it seems more important that we increase the accuracy of the classifier. That being said, previous results in this work would indicate that our selection of languages is poor. This is due to PTs subsuming both the RPF and LPM langauges. This makes it unsuprising that we achieved PT-like performance. We conjecture that a portfolio method with a greater number of learnable languages would produce higher accuracy models, mainly in scenarios where learning PTs with simulated annealing is ill-suited. Overall, we know from the ideal learning scheme that using a portfolio based preference learning system has the ability to perform at least as good as any of the single langauge learning methods we investigate.

# Chapter 7 Discussion

While each of the previous sections provides interesting avenues of research by themselves, when taken in aggregate they allow us to better understand various aspects of how to select a preference representation language and how a portfolio-based system would work for preference learning. This work looks at the use of a subsumption relation between languages, which has significant analytical power, but we mostly use it to determine which languages are best suited for learning. We also investigated various techniques for learning preferences and the potential for a single language to be used as a universal preference learner. Our data shows that while some languages may be usable in this capacity it is best to select the language used for learning on an ad hoc basis. To solve this problem of language selection we have chosen to build a portfolio preference learning system and test its viability.

## 7.1 Subsumption

Subsumption acts as an undercurrent throughout each of the previous sections. For example, learned PT models work well when learning example sets produced from RPF models. This relationship is implied by subsumption, since PTs subsume RPFs, and is supported by the data as well. We also see this when learning LPMs for example sets produced from some types of CLPM models. In both settings we see improved learning accuracy when a subsumption relation exists.

Subsumption provides a theoretical basis for the qualitative analysis and categorization of preference representation languages based on their expressive power. Our analysis is focused on the relatively limited set of static deterministic preferences over combinatorial domains, but we see no reason why the definition of subsumption cannot be extended. We propose two categories of preferences to which one might consider extending the definition of subsumption: probabilistic and dynamic preferences. There is a good deal of research into probabilistic preference models [Cornelio et al., 2013, Luce, 1958] making it a fertile field for potential subsumptive analysis. Dynamic preferences, preferences which change across time, are also of interest. Both of these extensions add significant features to preference representations and subsumption could help to better understand the structure of these more complex languages.

While extending subsumption to a broader class of languages is a good direction, there is still work to be done for our limited subset of languages. For one, we did not include all possible static deterministic languages, and as languages get added or modified it may not be possible to complete such analysis. Importantly, subsumption (and the base of languages we establish) shows us where a particular language fits based on its expressive power. This allows newly developed preference representation languages to better establish how they fit within the wider context of known preference languages.

## 7.2 Methods of Preference Learning

Local search has seen some success as a method for learning preferences [Allen et al., 2017] and we apply similar techniques to RPF and PT models. Alongside this we also investigate using artifical neural networks to learn qualitative preferences. Both of these methods learn approximations of the optimal preference model from their respective languages. We rely on approximations because the problem of learning an optimal model is NP-hard. This complexity barrier, assuming P $\neq$ NP, remains constant as we move from single-agent to multi-agent settings, even under the maximin criterion. While these problems cannot be directly solved in an efficient manner, our approximations show promising results.

While we establish the viability of using simulated annealing and neural networks for preference learning, our experiments uncovered another interesting result. Namely, learning a model from the same language as the model used to generate the example set maximizes learning accuracy. Thus it is beneficial to classify example sets based on preference language and this establishes the potential usefulness of a portfolio-based preference learning system.

Our work towards learning preferences can be furthered in several ways. Perhaps the easiest is expanding our tests, both in terms of the number of languages whose models are used to generate example sets and in terms of the languages whose models we can learn. Another extension is the application of simulated annealing to other preference languages. This would establish whether or not the technique of simulated annealing can be applied more generally, with specific focus on non-logical preference languages. Along these lines there are many hyperparameters in both simulated annealing and ANNs which can be modified to try and improve their accuracy.

## 7.3 Portfolio Preference Learning

Many of the individual pursuits in this work point to the best system for learning preferences being one where we first analyze a given example set and then select the best language for learning. This describes the portfolio method of solving the algorithm selection problem. We investigate a small scale version of such a system.

The first piece of evidence that portfolio preference learning is viable comes from the ability to describe subsumption relations between various preference representation languages. A language hierarchy based on how expressive power means that we might be able to restrict the number of learning algorithms we need, removing ones for redundant languages. Another piece of evidence supporting the portfolio approach comes from the learning algorithms we developed. Namely, if we can identify the language of the model that generated a given example set we can select the language which will have increased performance. Moreover, our learning data suggests this technique should work better than simply selecting a “universal” language, that is a language that is capable of representing a broad variety of preference orders.

We found that there are viable methods of classifying example sets based on the language of the model used to generate them. While not perfectly accurate, we found that using a feature vector based approach works well. This is in contrast to using GCNNs for the same classification task, whose accuracy was marginally above random chance.

Finally, we built a small portfolio-based preference learning system. Our results indicated that the portfolio approach has merit, but our system does not outperform learning a model from a sufficiently “universal” language, in our case a PT model. We see a need to further investigate not only our methods for classifying example sets, but also in the number and diversity of learning algorithms we have to use. Portfolio learning has the potential to be a very powerful preference learning tool, but requires further investigation and development.

# Chapter 8 Conclusion

In this work, we define and explore the relation between the languages used to represent preferences and the problem of preference learning. A foundational element of our analysis is the subsumption relation which forms between different preference representation languages. This relation can allow us to perform a number of useful analyses. The primary use of subsumption is to provide a qualitative comparison between pairs of languages which orders them based on how expressive they are. This induces a partial preorder giving us a hierarchy of languages. We build such a hierarchy for a limited set of languages which model preferences over combinatorial domains.

While subsumption promises to be a powerful tool for analyzing languages it cannot fully articulate the relationship between two languages. As an example of this we looked at the relationship between CP-nets and CLPMs. During our analysis we found that CLPMs can be used to approximate CP-nets, ensuring the production of a consistent, if stronger, preference relation. There exists a subset of CP-nets whose relations can be replicated with complete accuracy by CLPMs, which we describe in detail. Taken together these results show a tight correlation between CP-nets and CLPMs, while both languages are incomparable according to subsumption. This shows that subsumption should be used alongside other methods of analysis to best understand how two languages relate to one another.

We study the problem of preference learning by establishing the use of simulated annealing to learn RPF and PT models, and the use of ANNs to learn qualitative preferences. We found that both simulated annealing and ANNs produce sufficiently well performing models. In fact, both PTs and ANNs display a potential to be used as “universal” learning languages, that is contains models capable of learning many different types of example sets accurately. We also noticed a trend where learning accuracy is increased if the language used for learning was the same language as the one whose model generated the example set. This trend implies that the best course of action when learning preferences is to first determine which language that contains the model that generated a particular example set and then learn a model of that language.

In order to build such a learning system we first need a method of determining the language that contains the model which generated a particular example set. With this in mind we experimented with two ANN approaches (linear feed-forward ANNs and GCNNs) to find a method of classifying example sets. Our first approach consisted of manually engineering a feature vector based on basic information obtainable from an example set. While we found this produced decent classifiers there were still too many errors for our liking. While most of these errors were due to closely related preference languages we still wanted higher accuracy. We tried two methods for improving the accuracy of our classifier: extending the feature vector and using GCNNs. While extending our feature vector with new information did make a sizable improvement during training the accuracy was the same, or slightly worse, when we applied our models to unseen examples. Ultimately, we found that the best method for learning such classifiers was to use feature vectors as GCNNs performed very poorly, often not exceeding random chance and never exceeding the accuracy of our feature vector classifiers.

Using the knowledge gained throughout this work we built a small version of a portfolio-based preference learning system. What we found was that under ideal circumstances the approach of selecting the best language to learn a model from has experimental validity and is capable of producing much greater accuracy than a system which learns only a single language. That being said, our system performed similarly to learning a PT for any given example set, with some settings showing our system performing worse than the only PT system.

Ultimately, this work shows that there are two effective avenues to removing the language assumption from the problem of preference learning. The first is to choose the same language in all cases which performs well across a wide variety of example sets, for example PTs or ANNs. The second is to adapt the portfolio solution from the algorithm selection problem and build a portfolio-based preference learner which analyzes the example set and selects the best language to learn from a selected set of learnable preference representation languages. While we establish the validity of these two techniques additional testing is required. This body of work raises many questions and open problems, some narrow and some broad, but all fruitful places for future work.

## Bibliography

Sultan Ahmed and Malek Mouhoub. Constrained optimization with preferentially ordered outcomes. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 307–314. IEEE, 2018.

Eisa Alanazi, Malek Mouhoub, and Sandra Zilles. The complexity of learning acyclic CP-nets. In *IJCAI*, pages 1361–1367, 2016.

Thomas E Allen, Judy Goldsmith, Hayden Elizabeth Justice, Nicholas Mattei, and Kayla Raines. Generating CP-nets uniformly at random. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Thomas E Allen, Cory Siler, and Judy Goldsmith. Learning tree-structured CP-nets with local search. In *The Thirtieth International Flairs Conference*, 2017.

John Bartholdi, Craig A Tovey, and Michael A Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and welfare*, 6(2):157–165, 1989.

Peter Bartlett and Shai Ben-David. Hardness results for neural network approximation problems. In *European Conference on Computational Learning Theory*, pages 50–62. Springer, 1999.

Richard Booth, Yann Chevaleyre, Jérôme Lang, Jérôme Mengin, and Chattrakul Sombattheera. Learning various classes of models of lexicographic orderings. *PREFERENCE LEARNING*, page 1, 2009.

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

Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *Journal of Artificial Intelligence Research*, 21:135–191, 2004.

Michael Bräuning and Eyke Hüllermeier. Learning conditional lexicographic preference trees. *Archives of Data Science, Series A*, 1(1):41–55, 2016. ISSN 2363-9881. doi: 10.5445/KSP/1000058747/03.

Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. Answer set optimization. In *IJCAI*, volume 3, pages 867–872, 2003.

Vincent Conitzer, Jérôme Lang, and Lirong Xia. Hypercubewise preference aggregation in multi-issue domains. In *IJCAI proceedings-international joint conference on artificial intelligence*, volume 22, page 158, 2011.

Cristina Cornelio, Judy Goldsmith, Nicholas Mattei, Francesca Rossi, and K Brent Venable. Updates and uncertainty in cp-nets. In *Australasian Joint Conference on Artificial Intelligence*, pages 301–312. Springer, 2013.

Gerard Debreu. Representation of a preference ordering by a numerical function. *Decision processes*, 3:159–165, 1954.

Carmel Domshlak, Eyke Hüllermeier, Souhila Kaci, and Henri Prade. Preferences in AI: An overview. *Artificial Intelligence*, 175(7–8):1037–1052, 2011.

Didier Dubois, Henri Prade, and Fayçal Touazi. Conditional preference nets and possibilistic logic. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 181–193. Springer, 2013.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Peter C Fishburn. Lexicographic orders, utilities and decision rules: A survey. *Management science*, 20(11):1442–1471, 1974.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Marius Thomas Schneider, and Stefan Ziller. A portfolio solver for answer set programming: Preliminary report. In *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 352–357. Springer, 2011.

Edgard N Gilbert. Gray codes and paths on the n-cube. *The bell system technical journal*, 37(3):815–826, 1958.

Judy Goldsmith, Jérôme Lang, Miroslaw Truszczynski, and Nic Wilson. The computational complexity of dominance and consistency in CP-nets. *Journal of Artificial Intelligence Research*, 33:403–432, 2008.

L Habin and K Vlado. Combining mining of web server logs and web content for classifying users navigation pattern and predicting users future request. *Data Knowledge Eng*, 61(2006):304–330, 2007.

Peter Haddawy and Steve Hanks. Representations for decision-theoretic planning: Utility functions for deadline goals. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 71–82. Morgan Kaufmann Publishers Inc., 1992.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Michael Huelsman and Mirosław Truszczyński. Approximating dominance in CP-nets by lexicographic evaluation. In *Proceedings of the 6th International Conference on Algorithmic Decision Theory*, 2019.

Michael Huelsman and Mirosław Truszczyński. A lexicographic strategy for approximating dominance in CP-nets. In *The Thirty-Third International Flairs Conference*, 2020.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

Souhila Kaci. *Working with Preferences: Less is More*. Springer, 2011.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Rajeev Kohli and Kamel Jedidi. Representation and inference of lexicographic preference models and their variants. *Marketing Science*, 26(3):380–399, 2007.

Jérôme Lang and Lirong Xia. Sequential composition of voting rules in multi-issue domains. *Mathematical Social Sciences*, 57(3):304–324, 2009.

Jérôme Lang, Jérôme Mengin, and Lirong Xia. Voting on multi-issue domains with conditionally lexicographic preferences. *Artificial Intelligence*, 265:18–44, 2018.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, pages 1542–1543, 2003.

Xudong Liu and Mirosław Truszczyński. New complexity results on aggregating lexicographic preference trees using positional scoring rules. In *Proceedings of the 6th International Conference on Algorithmic Decision Theory*, 2019.

R Duncan Luce. A probabilistic theory of utility. *Econometrica: Journal of the Econometric Society*, pages 193–224, 1958.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

John R Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

Jörg Rothe, editor. *Economics and Computation: An Introduction to Algorithm Game Theory, Computational Scoial Choice, and Fair Division*. Springer, 2016.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Michael Schmitt and Laura Martignon. Complexity of lexicographic strategies on binary cues. *Preprint*, 302, 1999.

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

John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton university press, 1944.

Nic Wilson. Efficient inference for expressive comparative preference languages. In *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32: 565–606, 2008.

Fusun Yaman, Thomas J Walsh, Michael L Littman, and Marie Desjardins. Democratic approximation of lexicographic preference models. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1200–1207. ACM, 2008.

Ying Zhu. Preferences: Optimization, importance learning and strategic behaviors. 2016.

## Vita

- Name: Michael Andrew Huelsman
- Education:
  - Randall K. Cooper High School
    - 2008 - 2012
    - High School Diploma
    - Union, Kentucky
  - Transylvania University
    - 2012 - 2016
    - Bachelor's of Arts in Computer Science
    - Lexington, Kentucky
  - University of Kentucky
    - 2016 - 2021
    - Doctorate of Philosophy, Computer Science
    - Lexington, Kentucky
- Positions Held:
  - Teaching Assistant
    - CS215: Introduction to Program Design, Abstraction, and Problem Solving Techniques
      - Fall 2016, Spring 2017, Spring 2021
    - CS375: Logic and Theory of Computing
      - Fall 2020
  - Research Assistant
    - Summer 2016 - Summer 2020
- Publications:
  - Approximating dominance in CP-nets by lexicographic evaluation.
    M Huelsman and M Truszczynski - In Proceedings of the 6th International Conference on Algorithmic Decision Theory, 2019.
  - A Lexicographic Strategy for Approximating Dominance in CP-Nets
    M Huelsman, M Truszczynski - The Thirty-Third International Flairs Conference, 2020
  - Preference Aggregation over Combinatorial Domains Using Heuristic Search.
    M Huelsman and M Truszczynski - Presented the M-PREF Workshop at ECAI, 2020.

– The Role of Model Selection in Preference Learning
  M Huelsman, M Truszczynski - The Thirty-Fourth International Flairs Conference, 2021