MODELING, LEARNING AND REASONING ABOUT PREFERENCE TREES 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
Xudong Liu
Lexington, Kentucky

Director: Dr. Miroslaw Truszczynski, Professor of Computer Science
Lexington, Kentucky 2016

MODELING, LEARNING AND REASONING ABOUT PREFERENCE TREES OVER COMBINATORIAL DOMAINS

By

Xudong Liu

Director of Dissertation: Miroslaw Truszczynski

Director of Graduate Studies: Miroslaw Truszczynski

Date: April 4, 2016

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

Name	Date

DISSERTATION

Xudong Liu

The Graduate School

University of Kentucky

2016

MODELING, LEARNING AND REASONING ABOUT PREFERENCE TREES 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
Xudong Liu
Lexington, Kentucky

Director: Dr. Miroslaw Truszczynski, Professor of Computer Science
Lexington, Kentucky 2016

ACKNOWLEDGMENTS

I want to thank Dr. Truszczynski, other professors on my committee board, my colleagues, my parents, my wife and son, and my friends.

To my father Zhaoling Liu, my mother Ping Liu, my wife Xiaozhen Zhang, and my son Adam (Shude) Liu.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

Preferences are ubiquitous. They arise when we select ice-cream flavors, vote for candidates for an office, and buy cars. Preferences have spurred research in areas such as artificial intelligence, psychology, economics, and operations research. Specifically, there has been growing interest in research problems on preferences in contexts such as knowledge representation and reasoning, constraint satisfaction, decision making, and social choice theory. My research focuses on problems in preference modeling, reasoning and learning.

Preferences can be represented in a *quantitative* and *qualitative* manner. For the former, agents express preferences in a numerical form of a value function that precisely assesses the degree of satisfaction of objects (often called *outcomes* or *alternatives*). Specifying preferences as value functions on alternatives is feasible for humans in some situations, e.g., when the number of alternatives is limited. In other circumstances, particularly when the number of alternatives is large, people often cannot express their preferences directly and accurately as value functions [29].

Assume an agent is given three flavors of ice-cream: *strawberry*, *chocolate* and *vanilla*, and is asked to describe her preference among them. The agent could think of a value function that assigns quantities (*utilities*) to each outcome based on a scale from 1 to 10, with 10 representing the most satisfaction. For instance, the agent could give the following value function:

$$\textit{strawberry} \mapsto 6, \textit{chocolate} \mapsto 9 \text{ and } \textit{vanilla} \mapsto 3.$$

This function shows that the favorite alternative to the agent is *chocolate* (it has the highest utility), and *strawberry* is preferred over *vanilla*.

Instead of rating alternatives quantitatively, it is often easier and more intuitive

to give preferential information in a qualitative way, so that to specify a binary preference relation. Thus, the same agent could rank the flavors as the following preference order:

$$chocolate \succ strawberry \succ vanilla.$$

Note that one can obtain the qualitative preferences from the value function, but not vice versa. My research deals with qualitative preference relations.

Preferences play an essential role in areas that involve making decision, such as decision theory and social choice theory. Once we have preferences of a user or users, we can reason about the preferences to support decision making. In general, preference reasoning problems can be classified based on the number n of agents from which the preferences are gathered:

1. $n = 1$: individual decision making,
2. $n > 1$: collaborative decision making.

In case $n = 1$, we focus on optimization of the agent's preferences and help her make a better decision by, for example, computing an optimal alternative or comparing two given alternatives. For the case where $n > 1$, it is important to calculate a consensus (e.g., a winning outcome or ranking) of the group of agents.

One of the problems in preference reasoning is to aggregate preferences of a group of agents. The problem is central to collective decision making and has been studied extensively in social choice theory. Let us consider a scenario, where we are given a set of alternatives $X = \{a, b, c, d, e\}$ and a set P_X of 10 preferences (called *votes*) as follows.

$$5 : a > c > b > e > d,$$
$$3 : b > a > e > c > d,$$
$$2 : c > d > b > a > e.$$

Note that the integer associated with every preference order is the number of agents sharing that same preference. We are asked to compute the winning alternative according to some aggregation rule. Plurality, veto and Borda are examples of commonly used voting rules. For instance, Borda rule assigns score $m - i$ to the ith ranked alternative, where m is the number of alternatives. Thus, the winner is the alternative with the highest score. We compute that the Borda winner for P_X is a, since its score 31 is the highest, followed by candidates b and c with the second highest score 26.

While in the cases when the number of alternatives is small the preference-aggregating problems, such as dominance testing and winner determination, have received wide attention in the literature, the problems concerning preferences over *combinatorial domains*, which typically contain large numbers of outcomes, have not been investigated as much.

To illustrate the setting of combinatorial domains, let us consider a taxi company plans to purchase a fleet of cars. The features (or, as we will say, attributes) that will be taken into account are *BodyType*, *Capacity*, *Make*, *Price*, and *Safety*. Each attribute has a domain of values that it can take, e.g., *BodyType* may have four values *minivan*, *sedan*, *sport*, and *suv*. There could be hundreds or thousands of cars, described by different combinations of values on these five attributes, even for a relatively small number of attributes, and the decision makers will soon find it impossible to enumerate all of them from the most preferred to the least.

Consequently, an expressive yet concise representation is needed to specify preferences over combinatorial alternatives. Such preference formalisms are often categorized into *logical models* and *graphical models*. Logical models include penalty logic (*Pen-logic*) [46], possibilistic logic (*Poss-logic*) [30], qualitative choice logic (*Qual-logic*) [23], conditional preference theories (*CP-theories*) [76], and answer set optimization (*ASO*) [24], whereas graphical models found in the literature include gener-

alized additive independence networks (GAI-nets) [10, 45], lexicographic preference trees (*LP-trees*) [16, 56], conditional preference networks (*CP-nets*) [49], conditional preference networks with tradeoffs (*TCP-nets*) [18], and conditional importance networks (*CI-nets*) [49].

Once we fix a preference formalism, say $\mathcal{F}$, in which preferences of agents are specified, eliciting and learning preference expressions in $\mathcal{F}$ from agents becomes a fundamental problem. Different techniques have been proposed to preference learning in $\mathcal{F}$ such as *active learning* (or *preference elicitation*) and *passive learning* [37]. In the process of active learning, the algorithm iteratively asks the user for a pairwise preference between two given outcomes and constructs an instance of $\mathcal{F}$ as more preferences are elicited. For passive learning, the learning algorithm assumes that a set of pairwise preferences are obtained over a period of time and builds an instance of $\mathcal{F}$ with no more information from the user.

My research has centered around the language of LP-trees. Extending LP-trees, I have proposed two new tree-like preference formalisms: partial lexicographic preference trees (*PLP-trees*) [59] and preference trees (*P-trees*) [36, 58, 60]. The language of P-trees exploits a natural way humans apply to express preference information in the setting of combinatorial domains. Often a human agent would first consider the most desired *criterion*, possibly represented by a propositional formula φ. Outcomes that agree with it are preferred to those that do not. Then, the same mechanism is applied recursively to further discriminate among the outcomes that satisfy φ and among those that falsify φ. This process ends up with a structured preference system that always induces a total preorder.

My research on tree-like preference formalisms can be categorized into three main directions: preference modeling, preference learning and reasoning about preferences.

Preference Modeling My research formally proposed PLP-trees [59] and P-trees [58, 60]. In particular, I studied the relationship between P-trees and other existing

preference languages, and showed that P-trees extend LP-trees, possibilistic logic, and ASO rules. Moreover, my work established computational complexity results of commonly considered decision problems in the setting of P-trees, such as *dominance testing optimality testing*, and *optimality testing w.r.t a property*.

Preference Learning Given a set of pairwise preferences between alternatives, called *examples*, acquired from the user, it is important to learn (i) a PLP-tree, preferably of a small size, consistent with a dataset of examples, and (ii) a PLP-tree correctly ordering as many of the examples as possible in case of inconsistency. In my work, I studied both these problems [59]. I established complexity results for them and, in each case where the problem is in the class P, proposed a polynomial time algorithm. On the experimentation side, I have designed and implemented algorithms, using both Answer-Set Programming (ASP) and approximation methods, to learn PLP-trees and forests of these trees in the passive learning setting. To facilitate experimentation, I developed several datasets based on classification datasets such as *Library for Preferences*, *Preference Learning Site*, and *UCI Machine Learning Repository*. To evaluate the effectiveness and feasibility of our own models, I compared them with machine learning models, such as decision trees and random forests.

Preference Aggregation In this area, I investigated two preference-aggregation problems, the *winner* problem and the *evaluation* problem, based on *positional scoring rules* (such as k-approval and Borda) when votes in elections are given as LP-trees [55, 56]. My work brought new computational complexity results of these problems, and provided computational methods to model and solve the problems using *answer set programming* (ASP) and *weighted partial maximum satisfiability* (WPM).

The outline of the remainder of this thesis is the following. In Chapter 2, I present necessary technical preliminaries including binary relations, order theory, and computational complexity theory. In Chapter 3, I go through related work that proposed approaches to preference modeling and reasoning in artificial intelligence

and social choice theory. In Chapters 4 to 7, I discuss results of my work on modeling, learning and reasoning about preferences over combinatorial domains. I conclude with a brief note in Chapter 8 on my ongoing research, as well as on possible directions of future work.

Chapter 2 Technical Preliminaries

In this section I will give an overview of mathematical and computational concepts that I will use throughout the rest of this document. First, since I consider preference relations that are modeled as binary relations, I recall the definitions of binary relations and their key properties. I then define several types of preference relations in terms of these properties. Second, I review combinatorial domains as I am interested in preferences over combinatorial domains. Third, I introduce propositional logic to show how propositional formulas are used to compactly represent outcomes. Finally, I review concepts in computational complexity theory, as they are useful in describing the hardness of problems involving reasoning about preferences.

2.1 Relations and Orders

Definition 1. Let A and B be two sets of elements. A *binary relation* R between A and B is a subset of the Cartesian product of A and B, that is,

$$R \subseteq A \times B.$$

The following properties of binary relations are particularly relevant for modeling preferences.

Definition 2. Let R be a binary relation over a set O of objects ($R \subseteq O \times O$). We say that R is

1. reflexive if for every $o \in O$, $(o, o) \in R$.
2. irreflexive if for every $o \in O$, $(o, o) \notin R$.
3. total if for every $o_1, o_2 \in O$, $(o_1, o_2) \in R$ or $(o_2, o_1) \in R$.
4. transitive if for every $o_1, o_2, o_3 \in O$, if $(o_1, o_2) \in R$ and $(o_2, o_3) \in R$, then $(o_1, o_3) \in R$.

5. symmetric if for every $o_1, o_2 \in O$, if $(o_1, o_2) \in R$, then $(o_2, o_1) \in R$.
6. antisymmetric if for every $o_1, o_2 \in O$, if $(o_1, o_2) \in R$ and $(o_2, o_1) \in R$, then $o_1 = o_2$.

For instance, assuming that $\mathbb{N} = \{1, 2, \ldots\}$ is the set of positive integers, the *less-than-or-equal-to* relation $\leq$ over $\mathbb{N}$ is reflexive, total, transitive and antisymmetric, while the *less-than* relation $<$ over $\mathbb{N}$ is irreflexive, transitive and antisymmetric.

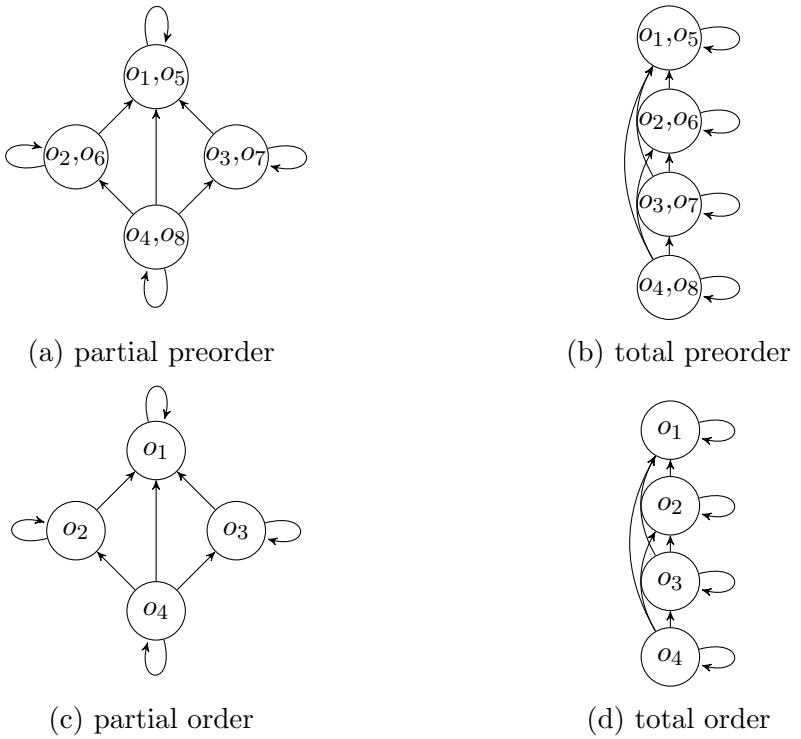
Definition 3. A binary relation over O is a *partial preorder* if it is reflexive and transitive, a *total preorder* if it is a partial preorder that is total, a *partial order* if it is a partial preorder that is antisymmetric, and a *total order* if it is a partial order that is total.

We use preorders to model preference relations. Thus, when we describe a preference order, we have in mind a relation that is a partial preorder. Given two objects o and o', we sometimes need to say that o' is at least as good as o or, that o' is strictly preferred over o. In some situations, due to lack of information about the two objects at hand, we cannot determine which object is preferred over the other, and speak about the objects being incomparable. Formally, we have the following definitions.

Definition 4. Let O be a set of objects, and o and o' two objects in O. Let $\succeq$ be a preference relation that is a partial preorder over O. We say that o' is weakly preferred to o if $o' \succeq o$. Object o' is strictly preferred to o, $o' \succ o$, if $o' \succeq o$ and $o \not\succeq o'$. Object o' is equivalent with o, $o' \approx o$, if $o' \succeq o$ and $o \succeq o'$. Object o' is incomparable with o, $o' \bowtie o$, if $o' \not\succeq o$ and $o \not\succeq o'$.

We illustrate these notions with several examples of preorders (preference orders) in Figure 2.1. We assume that a directed edge is from a less preferred object to a more preferred one. It is clear that the relation in Figure 2.1a is a partial preorder, Figure 2.1b a total preorder, Figure 2.1c a partial order, and Figure 2.1d a total

order. Note that in these figures, each node represents a set of distinct but equivalent outcomes.



(a) partial preorder (b) total preorder

(c) partial order (d) total order

Figure 2.1: Binary relations

Definition 5. Let R and R' be two binary relations, R' *extends* R if $R \subseteq R'$.

As an example of relation extensions, we consider the partial order $\succeq$ in Figure 2.1c. Since $o_2 \bowtie o_3$, we have in total two extensions:

$$o_1 \succeq o_2 \succeq o_3 \succeq o_4,$$
$$o_1 \succeq o_3 \succeq o_2 \succeq o_4.$$

Definition 6. Let $\succeq$ be a preference relation over O, $o \in O$ is *optimal* if there does not exist $o' \in O$ such that $o' \succ o$.

For instance, object o_1 is optimal in the partial preorder shown in Figure 2.1a.

2.2 Combinatorial Domains

One scenario when decision problems involving preferences are difficult is when outcomes are described as combinations of attribute values from finite domains. Take the domain of cars as an example, where the attributes we care about are *Price*, *Safety*, and *Capacity*. Every attribute has a binary domain of values: *Capacity* with domain {*low,high*}, *Price* with domain {*low,high*}, and *Safety* with domain {*low,high*}. Note that, although we mostly use binary attributes in the thesis, the results and algorithms we have obtained can easily be adjusted to general non-binary cases. Since these are the only aspects of a car we care about, cars can be described as vectors of values from these domains. For instance, vector $\langle$*high,low,high*$\rangle$ represents a car that has high capacity, low price and high safety. We clearly see that the number of cars grows exponentially as there are more attributes.

Definition 7. Let $\mathcal{I}$ be a set of attributes $\{X_1, \ldots, X_p\}$, each attribute X_i associated with a finite domain $Dom(X_i)$. A *combinatorial domain* $CD(\mathcal{I})$ is a set of combinations of values from $Dom(X_i)$:

$$CD(\mathcal{I}) = \prod_{X_i \in \mathbf{V}} Dom(X_i).$$

We call the elements of $CD(\mathcal{I})$ outcomes. Clearly, the size of $CD(\mathcal{I})$ is exponential in p, the number of attributes. The exponential growth of $|CD(\mathcal{I})|$ makes it hard, if not impossible, for agents to directly assess their preferences, even when each domain is binary and there are as few as 6-7 attributes. In many practical cases, hard constraints that can be modeled, for instance, by propositional formulas, are identified and imposed to eliminate the infeasible outcomes.

2.3 Propositional Logic

In this work, *propositional logic* plays an important role in compactly representing preferences over combinatorial domains. Propositional logic [47], or propositional

calculus, is a logic language concerning propositions (e.g., statements that are true or false) that are built upon atomic propositions by means of logical connectives. We first define the syntax of the language, that is, how formulas are constructed. Then, we show its semantics, i.e., what it means for a formula to be true or false.

Propositions are represented as *well-formed formulas*, or simply *formulas* when no ambiguity. Formulas are built from an alphabet of truth symbols ($\top$ and $\bot$), variables (uppercase letters), connectives ($\neg$, $\wedge$, $\vee$, and $\rightarrow$), and parantheses. A *formula* is either a truth symbol, a variable, or, if φ and ψ are formulas, $(\neg\varphi)$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, and $(\varphi \rightarrow \psi)$. (Outside pair of parentheses are often left out. In addition, conventions based on the binding strength of connectives are used to eliminate some other pairs of parentheses.) For example, if X and Y are formulas, we have that $X \wedge (X \rightarrow \neg Y)$ is a formula.

A *truth assignment* is a mapping v from variables to logical values T (true) and F (false). We now define what it means for a truth assignment to *satisfy* and *falsify* a formula. Let v be a truth assignment, X a variable, and φ and ψ propositional formulas. First, we define that v satisfies X, denoted by $v \models X$, if $v(X) = T$; and that v falsifies X, denoted by $v \not\models X$, if $v(X) = F$. Then, we have $v \models \neg\varphi$ if $v \not\models \varphi$ holds, $v \models \varphi \wedge \psi$ if $v \models \varphi$ and $v \models \psi$ hold, $v \models \varphi \vee \psi$ if $v \models \varphi$ or $v \models \psi$ holds, $v \models \varphi \rightarrow \psi$ if $v \not\models \varphi$ holds or $v \models \psi$ holds. We always have $v \models \top$ and $v \not\models \bot$.

A truth assignment can then be viewed as an outcome in a combinatorial domain. We consider two types of combinatorial domains: those of *binary* attributes and those of *non-binary* attributes.

For a combinatorial domain of binary attributes, such attributes correspond to variables in a language of propositional logic, and the outcomes from such a combinatorial domain correspond to truth assignments for that language. To establish the correpondence, it suffices to select one value in the domain of each attribute as *True* and the other one as *False*. Thus, propositional formulas provide a convenient way of

representing sets of, possibly exponentially many, outcomes from the corresponding combinatorial domain. We consider the domain of cars as discussed in Section 2.2. We write variables C, P and S in propositional logic, corresponding to attributes *Capacity*, *Price*, and *Safety*. We then set that the variables C, P and S being *True* represent *high* in the attributes *Capacity*, *Price* and *Safety*, repectively. Immediately, the variables being *False* means *low* in the attributes. As a consequence, formula $C \wedge \neg P$ is a shorthand for the set of cars that have high capacities and low prices.

When the attributes in the combinatorial domain become in general non-binary, we now view the *values* in the attribute domains, not the attributes, as variables in propositional logic. A variable assigned *True* (*False*) in M means corresponding attribute value is in o (is not in o, respectively). Then, a formula φ represents the set of outcomes whose counterpart truth assignments satisfying φ. We look at the domain of cars of non-binary attributes: *Capacity* with domain {*2,5,7m*}, *Price* with domain {*low,med,high,vhigh*}, and *Safety* with domain {*low,med,high*}. This domain corresponds to a language of propositional logic of 10 variables, because there are 10 attribute values. We denote these variables by T_C, F_C, S_C, L_P, M_P, H_P, V_P, L_S, M_S and H_S, in order of the values in attributes *Capacity*, *Price*, and *Safety*. A truth assignment, that sets *True* on variables S_C, M_P, and H_S, and *False* on the others, models a car that has small capacity, medium price and high safety. We see that not all truth assignments are legal. Therefore, we need a constraint that, for every attribute domain, exactly one variable is true. Such a constraint can be expressed as a propositional formula Φ. Now it is clear that formula $\Phi \wedge ((H_P \wedge S_C) \vee (M_P \wedge M_S))$ precisely and concisely represents the set of cars that have high price and capacity of 7 or more, and cars that have medium price and medium security. Thus, all results we have obtained for combinatorial domains over binary attributes apply to the general case, too.

2.4 Computational Complexity Theory

Computer scientists looking for algorithms to solve computational problems seek ways to classify problems according to their computational hardness in terms of time (the number of instructions needed to solve the problem) or space (the size of memory needed to solve the problem). In this section, we define classes of computational complexity used for such classification. We assume familiarity with the concept of the *Turing machine* (TM). The definition of this notion and other definitions discussed below can be found in complexity books by Garey and Johnson [38]; Lewis and Papadimitriou [54]; and Arora and Barak [7].

Decision Problems

Let Σ be a finite set of elements. A *string* over alphabet Σ is an ordered tuple of finite elements from Σ. In complexity theory, Σ is typically binary, that is, $\Sigma = \{0, 1\}$. We denote by Σ^* the set of all strings of elements in Σ. A *decision problem* (or a *language*) is a set L of strings such that $L \subseteq \Sigma^*$. For instance, the SAT problem is the set of all finite propositional formulas that have a satisfying truth assignment (assuming some natural representation of propositional formulas as strings over a finite alphabet).

Studying decision problems on preferences involves designing reasoning algorithms and proving complexity results. Hence, it is important to review complexity classes that are related to later discussions of computational complexity results. These classes include **P**, **NP**, **coNP**, classes in the polynomial hierarchy, and **PSPACE**.

P, NP and coNP

What differentiates the two classes **P** and **NP** is whether the decision problem can be solved by a deterministic or a nondeterministic TM. [7]

Let $f(n)$ be the computation time to solve a problem of input size n. We denote by **DTIME**$(f(n))$ (**NTIME**$(f(n))$) a set of decision problems for which there exists

a deterministic (nondeterministic, respectively) TM that solves any instance of the problem in time $f(n)$. We now define the two classes as follows.

Definition 8 (Garey and Johnson, 1979)**.** The class **P** (**NP**) consists of the decision problems that can be solved using a deterministic (nondeterministic, respectively) TM in time polynomial in the size of the input. Formally, we have

$$\mathbf{P} = \bigcup_{d\in\mathbb{N}} \mathbf{DTIME}(n^d),$$
$$\mathbf{NP} = \bigcup_{d\in\mathbb{N}} \mathbf{NTIME}(n^d),$$

where n is the size of the input.

Researchers in the field of complexity theory have studied the relation between these two classes. Clearly, the relation $\mathbf{P} \subseteq \mathbf{NP}$ holds. Whether $\mathbf{NP} \subseteq \mathbf{P}$ holds or not remains an open question. However, it is strongly believed that $\mathbf{P} \neq \mathbf{NP}$ [40].

One of the many complexity classes related to **P** and **NP** [40] is the class **coNP**, which contains problems that are complements of the problems in **NP**. Let $L \subseteq \{0,1\}^*$ be a decision problem, we denote by $\overline{L}$ the complement of L, that is, $\overline{L} = \{0,1\}^* - L$. We have the following definition of the class **coNP**.

Definition 9. $\mathbf{coNP} = \{L : \overline{L} \in \mathbf{NP}\}$.

To characterize the most difficult problems in class C (**NP**, **coNP**, etc), it is helpful to introduce the definition of polynomial-time reducibility [40] and the idea of C-hardness.

Definition 10. A decision problem $L \subseteq \{0,1\}^*$ is *polynomial-time reducible* to a decision problem $L' \subseteq \{0,1\}^*$, $L \leq_p L'$, if there is a polynomial-time computable function $g : \{0,1\}^* \to \{0,1\}^*$ such that for every instance $x \in L$ *iff* $g(x) \in L'$. If C is a class of decision problems, we say that L' is *C-hard* if $L \leq_p L'$ for every L in class C.

Definition 11. Let C be a complexity class (**NP**, **coNP**, etc). A decision problem L' is *C-complete* if L' is in the class C and L' is C-hard.

It is clear that, in order to prove C-completeness, one needs to show that $L' \in C$ (membership of class C), and prove C-hardness.

TM with Oracles and Polynomial Hierarchy

A *TM with an oracle* for a decision problem L is a TM that makes calls to an oracle that decides L. The *polynomial hierarchy*, denoted by **PH**, is a hierarchy of these complexity classes (i.e., Δ_i^P, Σ_i^P, and Π_i^P) that generalize the classes **P**, **NP** and **coNP** to oracles.

Definition 12. The **PH** is defined iteratively. We first define that $\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathbf{P}$. Then for $i \geq 0$, we define Δ_{i+1}^P (Σ_{i+1}^P) to consist of decision problems solvable by a polynomial-time deterministic (nondeterministic, respectively) TM with an oracle for some Σ_i^P-complete problem. We denote by Π_{i+1}^P the set of decision problems that are complements of problems in Σ_{i+1}^P.

For example, Σ_2^P is the class of decision problems solvable by a nondeterministic TM in polynomial time with an oracle for some **NP**-complete problem.

One may notice that the classes Σ_i^P and Π_i^P consist of problems that are complements to each other. Moreover, we have the inclusion between these classes as shown in Figure 2.2.

PSPACE

In this work, we consider yet another complexity class called **PSPACE** that concerns the complexity of space. It consists of problems that can be decided in polynomial space.

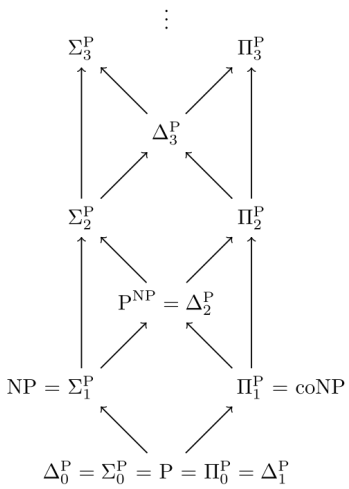


Figure 2.2: Polynomial hierarchy diagram

Definition 13. The class **PSPACE** is the class of decision problems solvable by a TM in space polynomial in the size of the input.

It is not hard to see the following relation hold.

$$\mathbf{PH} \subseteq \mathbf{PSPACE}.$$

We illustrate the relationship among the complexity classes in Figure 2.3. Many classes that are not in our focus are omitted from our diagram.

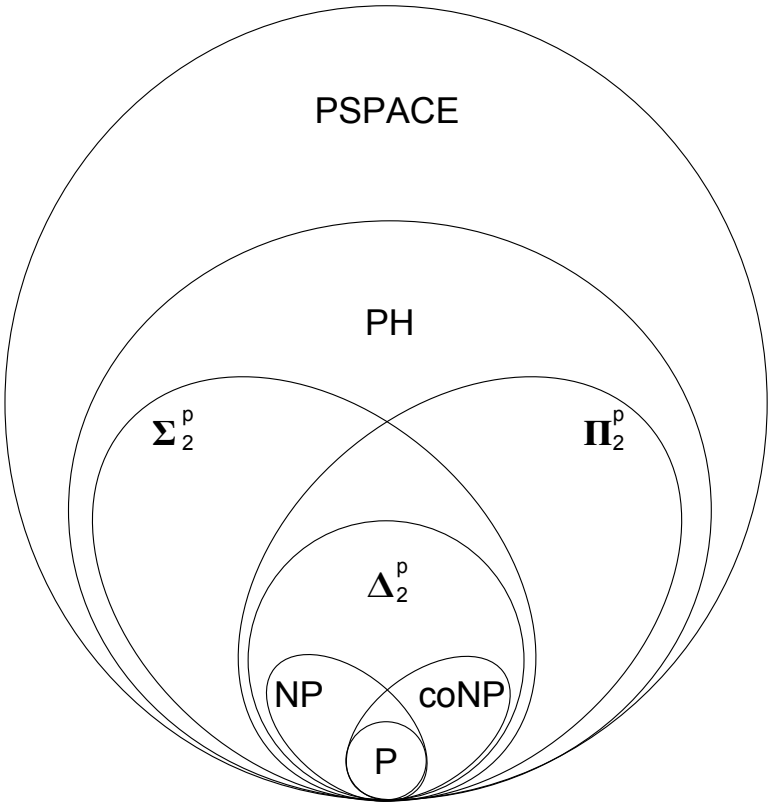


Figure 2.3: Computational complexity diagram

Chapter 3 Related Work

In this section I will present work related to my thesis research. I will first review some of the preference systems that were introduced before, designed to represent qualitative preferences over combinatorial domains. I will then introduce concepts from social choice theory underlying methods to combine individual preferences to reach a common decision.

3.1 Preferences Modeling and Reasoning

Researchers have proposed several languages to model preferences. I will now discuss those of them that are closely related to my work. These languages include graphical formalisms: *Conditional Preference Networks* (CP-nets) and *Lexicographic Preference Trees* (LP-trees); and logical formalisms: *Possibilistic Logic* and *Answer Set Optimization* (ASO). They are developed to provide concise and intuitive presentations of preferential information for objects from combinatorial domains.

For all systems, I will focus on two aspects: the language in which preferences are specified, and complexity of and algorithms for problems about the model. The most fundamental of these problems are introduced in the following definitions.

Definition 14. $\mathcal{L}$-CONSISTENCE: given an instance $\mathcal{C}$ of a preference formalism $\mathcal{L}$, decide whether $\mathcal{C}$ is consistent, that is, whether there exists a total order of outcomes that agrees with every preference statement in $\mathcal{C}$.

Definition 15. $\mathcal{L}$-DOMINANCE: given an instance $\mathcal{C}$ of a preference formalism $\mathcal{L}$ and its two distinct outcomes o_1 and o_2, decide whether $o_1 \succ_{\mathcal{C}} o_2$, that is, whether o_1 is strictly preferred to o_2 in the preference order defined by $\mathcal{C}$.

Definition 16. $\mathcal{L}$-OPTIMALITY-I: given an instance $\mathcal{C}$ of a preference formalism $\mathcal{L}$, decide whether $\mathcal{C}$ has an optimal outcome.

Definition 17. $\mathcal{L}$-OPTIMALITY-II: given an instance $\mathcal{C}$ of a preference formalism $\mathcal{L}$ and an outcome o of $\mathcal{C}$, decide whether o is an optimal outcome.

Definition 18. $\mathcal{L}$-OPTIMALITY-III: given an instance $\mathcal{C}$ of a preference formalism $\mathcal{L}$ and some property Φ expressed as a Boolean formula over the alphabet of $\mathcal{C}$, decide whether there is an optimal outcome o that satisfies Φ.

Conditional Preference Networks

The Language. Conditional Preference Networks (CP-nets) define preferential relations between outcomes based on the *ceteris paribus* semantics [17]. *Ceteris paribus* is Latin for "everything else being equal."

Let $\mathbf{V}$ be a set of binary attributes.[1] We denote by $Asst(\mathbf{V})$ the set of all truth assignments to the attributes in $\mathbf{V}$. For each attribute $X_i \in \mathbf{V}$, $Pa(X_i)$ denotes the *parent* attributes of X_i, such that preferences over the domain of X_i depend upon how $Pa(X_i)$ are evaluated.

Definition 19. Let $\mathbf{V}$ be a set of binary attributes $\mathbf{V} = \{X_1, \ldots, X_n\}$. A CP-net over $\mathbf{V}$ is a tuple (G,T), where

1. $G = (V, E)$ is a directed graph, also called a *dependency graph*, specifying dependencies among attributes; an arrow in the dependency graph points to a child attribute from a parent attribute; for every $X_i \in V$, we have $Pa(X_i) = \{X_j : (X_j, X_i) \in E\}$; and
2. T is a collection of conditional preference tables (CPTs) for all attributes. A $CPT(X_i)$ consists of preference statements of the form

[1] Attributes in CP-nets can be multi-valued. However, as my research mostly deals with preference models over binary attributes, it suffices to discuss CP-nets in the binary setting.

$$\mathbf{u} : \succ^i_{\mathbf{u}},$$

where $\mathbf{u} \in Asst(Pa(X_i))$ and $\succ^i_{\mathbf{u}}$ is a total order describing preferences over $Dom(X_i)$ for a given assignment u to $Pa(X_i)$.

We say that a CP-net $N = (G, T)$ is *acyclic* if G is acyclic; otherwise, it is *cyclic*. To illustrate, let us consider the domain of cars. For simplicity, we take three binary attributes *Price*, *Safety*, and *Capacity*. Attribute *Capacity* (C) has two values *high* (c) and *low* ($\bar{c}$). Attribute *Price* (P) has two values *high* (p) and *low* ($\bar{p}$). Attribute *Safety* (S) has two values *high* (s) and *low* ($\bar{s}$). An example CP-net $N = (G, T)$ over binary attributes $\mathbf{V} = \{C, P, S\}$ is shown in Figure 3.1a. We see that the preferences on *Price* (*Safety*) depend upon the assignment made to *Capacity* (*Price*, respectively).

To decide if outcome o_1 is preferred to outcome o_2 in a CP-net N, one needs to show that o_2 can be improved, ceteris paribus, according to the preference statements in N to reach o_1.

Definition 20. Let N be a CP-net over $\mathbf{V}$, $X_i \in \mathbf{V}$, $\mathbf{U} = Pa(X_i)$, and $\mathbf{Y} = \mathbf{V} - (\mathbf{U} \cup \{X_i\})$. Let $\mathbf{u}x_i\mathbf{y}$ be an outcome, where $x_i \in Dom(X_i)$, $\mathbf{u} \in Asst(\mathbf{U})$, and $\mathbf{y} \in Asst(\mathbf{Y})$. An *improving flip* of $\mathbf{u}x_i\mathbf{y}$ wrt X_i is an outcome $\mathbf{u}x'_i\mathbf{y}$ such that $x'_i \succ^i_{\mathbf{u}} x_i$. A *sequence of improving flips* wrt N is a sequence of outcomes $o_1, \ldots, o_j$ such that, for every $k < j$, o_{k+1} is an improving flip of o_k wrt some attribute in $\mathbf{V}$. We say that outcome o_1 is preferred to outcome o_2 in N, denoted by $o_1 \succ_N o_2$, if there exists a sequence of improving flips from o_2 to o_1.

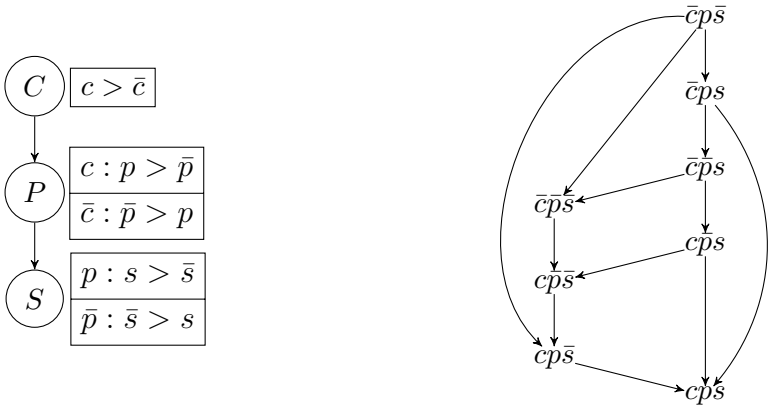
In a CP-net N, we say that outcome o is *optimal* if there does not exist another outcome o' such that $o' \succ_N o$.

Consider the CP-net N in Figure 3.1. It induces a partial order shown in Figure 3.1b, where each arrow represents an improving flip between two outcomes. We see that $cp\bar{s} \succ_N \bar{c}ps$ because of the improving flipping sequence: $cp\bar{s}, \bar{c}p\bar{s}, \bar{c}\bar{p}\bar{s}, \bar{c}\bar{p}s, \bar{c}ps$. Outcome cps is optimal because no other outcome is better. We note that there is no

flipping sequence between $c\bar{p}s$ and $\bar{c}\bar{p}\bar{s}$. In such case, we say that the two outcomes are *incomparable*. This CP-net is consistent because there exists a total order of outcomes that agrees with the preference graph Figure 3.1b. There are in fact two such total orders:

$$cps \succ cp\bar{s} \succ c\bar{p}\bar{s} \succ \boldsymbol{c\bar{p}s} \succ \boldsymbol{\bar{c}\bar{p}\bar{s}} \succ \bar{c}\bar{p}s \succ \bar{c}ps \succ \bar{c}p\bar{s},$$

$$cps \succ cp\bar{s} \succ c\bar{p}\bar{s} \succ \boldsymbol{\bar{c}\bar{p}\bar{s}} \succ \boldsymbol{c\bar{p}s} \succ \bar{c}\bar{p}s \succ \bar{c}ps \succ \bar{c}p\bar{s}.$$



(a) Dependency graph and CPT's (b) Preference graph

Figure 3.1: Acyclic CP-net

Problems and Complexity. Boutilier, Brafman, Domshlak, Hoos and Poole [17] have proved that every acyclic CP-net is consistent, whereas Goldsmith, Lang, Truszczynski and Wilson [44] have shown that the CPN-CONSISTENCE problem is PSPACE-complete in general.

For the CPN-DOMINANCE problem, its complexity depends on the structure of the dependency graph. The CPN-DOMINANCE problem can be solved by a polynomial time algorithm for binary-valued tree-structured CP-nets, and the problem is NP-complete for binary-valued CP-nets with specially structured dependency graphs (e.g., max-δ-connected dependency graphs) [17]. However, it is NP-hard for general binary-valued acyclic CP-nets [17]. Furthermore, in the most general case when the

dependency graph could be cyclic, this problem is PSPACE-complete even if the CP-nets are consistent [44].

For acyclic CP-nets, the optimality problems (i.e., CPN-OPTIMALITY-I, CPN-OPTIMALITY-II, and CPN-OPTIMALITY-III) are easy [17].

Lexicographic Preference Trees

The language of lexicographic preference trees [16] uses trees to model preferences. It is motivated by lexicographic orderings [49] and lexicographic preferences [34]. This formalism and its variants are the primary focus on my research.

The Language. *A lexicographic preference tree* (*LP-tree*) T over a set $\mathcal{I}$ of p binary attributes $X_1, \ldots, X_p$ is a labeled *binary tree*. Each node t in T is labeled by an attribute from $\mathcal{I}$, denoted by $Iss(t)$, and with *preference information* of the form $a > b$ or $b > a$ indicating which of the two values a and b comprising the domain of $Iss(t)$ is preferred (in general the preference may depend on the values of attributes labeling the ancestor nodes). We require that each attribute appears exactly once on each path from the root to a leaf.

Intuitively, the attribute labeling the root of an LP-tree is of highest importance. Alternatives with the preferred value of that attribute are preferred over outcomes with the non-preferred one. The two subtrees refine that ordering. The left subtree determines the ranking of the preferred "upper half" and the right subtree determines the ranking of the non-preferred "lower half." In each case, the same principle is used, with the root attribute being the most important one. We note that the attributes labeling the roots of the subtrees need not be the same (the relative importance of attributes may depend on values for the attributes labeling the nodes on the path to the root).

The precise semantics of an LP-tree T captures this intuition. Given an outcome $x_1x_2 \ldots x_p$, we find its preference ranking in T by traversing the tree from the root

to a leaf. When at node t labeled with the attribute X_i, we follow down to the left subtree if x_i is preferred according to the preference information at node t. Otherwise, we follow down to the right subtree.

It is convenient to imagine the existence of yet another level of nodes in the tree, not represented explicitly, with each node in the lowest level "splitting" into two of these implicit nodes, each representing an outcome. Descending the tree given an outcome in the way described above takes us to an (implicit) node that represents precisely that outcome's rank. The more to the left the node representing the outcome, the more preferred it is, with the one in the leftmost (implicit) node being the most desirable one as left links always correspond to preferred values.

To illustrate these notions, let us consider an example LP-tree over the car domain, given by the three binary attributes *Capacity*, *Price*, and *Safety*, described as earlier. Our agent prefers cars with high capacity to cars with low capacity, and this preference on *Capacity* is the most important one. Then, for high-capacity cars, the next most important attribute is *Safety* and she prefers cars with high security level, and the least important attribute is *Price*. She prefers low-price cars if security is low, and high-price, otherwise. For low-capacity cars, the importance of *Safety* and *Price* changes with *Price* being more important. The agent prefers low-price cars among the low-capacity. Finally, high-security cars are preferred over low-security cars. These preferences are captured by the LP-tree T in Figure 3.2. The tree shows that the most preferred car for our agent has high capacity, security, and price, and the next in order of preference has high capacity and security but low price.

Sometimes LP-trees can be represented in a more concise way. For instance, if for some node t, its two subtrees are identical (that is, the corresponding nodes are assigned the same attribute), they can be collapsed to a single subtree, with the same assignment of attributes to nodes. To retain preference information, at each node t' of the subtree we place a *conditional preference table*, and each preference in it

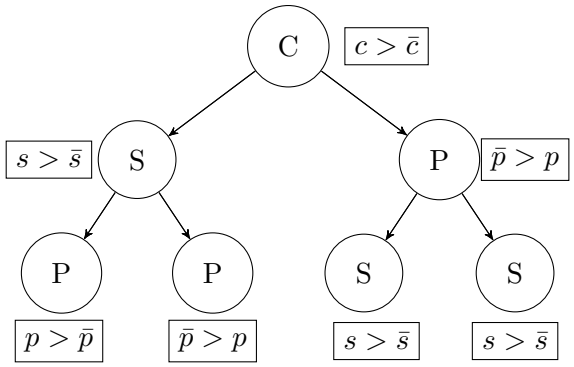


Figure 3.2: An LP-tree T

specifies the preferred value for the attribute labeling that node given the value of the attribute labeling t. In the extreme case when for every node its two subtrees are identical, the tree can be collapsed to a path.

Formally, given an LP-tree (possibly with some subtrees collapsed), for a node t, let $NonInst(t)$ be the set of ancestor nodes of t whose subtrees were collapsed into one, and let $Inst(t)$ represent the remaining ancestor nodes. A *parent* function $\mathcal{P}$ assigns to each node t in T a set $\mathcal{P}(t) \subseteq NonInst(t)$ of *parents* of t, that is, the nodes whose attributes may have influence on the local preference at $Iss(t)$. Clearly, the conditional preference table at t requires only $2^{|\mathcal{P}(t)|}$ rows, possibly many fewer than in the worst case. In the extreme case, when an LP-tree is a path and each node has a bounded (independent of p) number of parents, the tree can be represented in $O(p)$ space.

If for every node t in an LP-tree, $\mathcal{P}(t) = \emptyset$, all (local) preferences are unconditional and conditional preference tables consist of a single entry. Such trees are called *unconditional preference* LP-trees (UP trees, for short). Similarly, LP-trees with all non-leaf nodes having their subtrees collapsed are called an *unconditional importance* LP-trees (UI trees, for short). This leads to a a natural classification of LP-trees into four classes: unconditional importance and unconditional preference LP-trees (UI-IP trees), unconditional importance and conditional preference trees (UI-CP trees), etc.

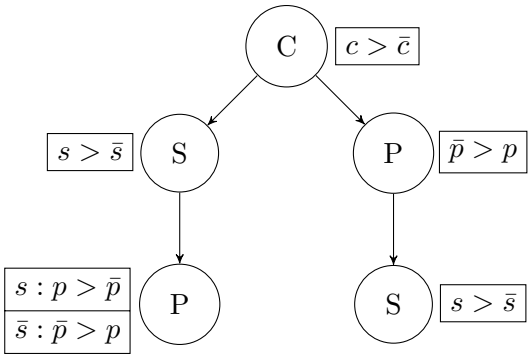


Figure 3.3: A CI-CP LP-tree T

The class of CI-CP trees comprises all LP-trees, the class of UI-UP trees is the most narrow one.

The LP-tree T in Figure 3.2 can be represented more concisely as a (collapsed) CI-CP tree v in Figure 3.3. Nodes at depth one have their subtrees collapsed. In the tree in Figure 3.2, the subtrees of the node at depth 1 labeled P are not only identical but also have the same preference information at every node. Thus, collapsing them does not incur growth in the size of the conditional preference table.

An LP-tree consisting of p binary attributes corresponds to a total order over 2^p outcomes. For the example in Figure 3.3, the total order induced by T is

$$cps \succ c\bar{p}s \succ c\bar{p}\bar{s} \succ cp\bar{s} \succ \bar{c}p\bar{s} \succ \bar{c}\bar{p}\bar{s} \succ \bar{c}ps \succ \bar{c}p\bar{s}.$$

Problems and Complexity. As any LP-tree induces a total order, the LP-CONSISTENCE problem is trivial. Moreover, an optimal outcome always exists and the LP-OPTIMALITY-I problem is trivial, too. Similarly, the LP-OPTIMALITY-II and LP-OPTIMALITY-III problems are easy to solve. Deciding whether outcome o_1 dominates outcome o_2 is done by traversing the tree until an attribute X is reached such that $o_1(X) \neq o_2(X)$. Alternatives o_1 and o_2 are then ordered based on the preference information on X [16]. This method works in polynomial time and so, we know that the LP-DOMINANCE problem is in **P**.

In addition to problems of reasoning about a single LP-tree, recently researchers

have initiated studies of the problem of aggregating LP-trees expressing preferences of multiple agents. The goal is to facilitate collaborative decision making. LP-trees are aggregated according to some social choice scheme, such as issue-by-issue voting [32], sequential majority voting rule [77], positional scoring rules (e.g. Borda, k-Approval) [52, 57]. Basics of social choice are discussed later in this chapter. In Chapter 7, I will provide detailed definitions of aggregating problems and results I obtained on their complexity according to positional scoring rules, as well as experimental analysis of computational methods I proposed for aggregating votes given as LP-trees. These methods are based on answer-set programming (ASP) [42] and weighted partial maximum satisfiability (WPM) [67].

Possibilistic Logic

Possibilistic logic [30] describes atomic preferences as weighted propositional formulas and uses collections of weighted formulas to specify preference relations.

The Language and the Model. A possibilistic logic theory Π over a vocabulary $\mathcal{I}$ is a set of *preference pairs*

$$\{(\phi_1, a_1), \ldots, (\phi_m, a_m)\},$$

where every ϕ_i is a propositional formula over $\mathcal{I}$, and every a_i is a real number such that $1 \geq a_1 > \ldots > a_m \geq 0$ (if two formulas have the same weight, they can be replaced by their conjunction). Intuitively, a_i represents the importance of ϕ_i, with larger values indicating higher importance.

The *tolerance degree* of an outcome o with regard to a preference pair (ϕ, a), $TD_{(\phi,a)}(o)$, is defined by

$$TD_{(\phi,a)}(o) = \begin{cases} 1, & o \models \phi \\ 1 - a, & o \not\models \phi \end{cases}$$

Based on that, the tolerance degree of an outcome o with regard to a theory Π of preference pairs, $TD_\Pi(o)$, is defined by

$$TD_\Pi(o) = min\{\, TD_{(\phi_i,a_i)}(o) : 1 \leq i \leq m\}.$$

The larger $TD_\Pi(o)$, the more preferred o is; that is, given two outcomes o_1 and o_2, we have

$$o_1 \succ_\Pi o_2 \textit{ iff } TD_\Pi(o_1) > TD_\Pi(o_2),$$
$$o_1 \approx_\Pi o_2 \textit{ iff } TD_\Pi(o_1) = TD_\Pi(o_2).$$

Intuitively, the most preferred outcomes are those satisfying all the formulas in Π. The next preferred ones are those that falsify some formulas $\phi_i \in \Pi$, the smallest a_m of which is maximal.

Let us look at the combinatorial domain of cars over three binary attributes *Capacity*, *Price*, and *Safety*. with values *high* (c) and *low* ($\bar{c}$), *high* (p) and *low* ($\bar{p}$), and *high* (s) and *low* ($\bar{s}$), respectively. Consider that an agent presents the following possibilistic theory P with two preference pairs.

$$P = \{(c \wedge s, 0.8), (s \Leftrightarrow p, 0.6)\}.$$

Intuitively, the agent expresses the following preferences. She likes the most the cars that falsifies neither $c \wedge s$ nor $s \Leftrightarrow p$. Her next preferred cars are those falsifying $s \Leftrightarrow p$, but satisfying $c \wedge s$. The least preferred cars for her are the ones falsifying $c \wedge s$. We now compute the tolerance degrees of outcomes with regard to P and show the results in Table 3.1. The theory P, thus, induces a total preorder:

$$cps \succ cp\bar{s} \succ c\bar{p}s \approx c\bar{p}\bar{s} \approx \bar{c}ps \approx \bar{c}p\bar{s} \approx \bar{c}\bar{p}s \approx \bar{c}\bar{p}\bar{s}.$$

Answer Set Optimization

The formalism of Answer Set Optimization (ASO) was introduced by Brewka, Niemelä and Truszczynski [25] and later enhanced by Brewka [22].

Outcomes	$TD_{(\phi_1,a_1)}(o)$	$TD_{(\phi_2,a_2)}(o)$	$TD_P(o)$
cps	1	1	1
$cp\bar{s}$	0.2	0.4	0.2
$c\bar{p}s$	1	0.4	0.4
$c\bar{p}\bar{s}$	0.2	1	0.2
$\bar{c}ps$	0.2	1	0.2
$\bar{c}p\bar{s}$	0.2	0.4	0.2
$\bar{c}\bar{p}s$	0.2	0.4	0.2
$\bar{c}\bar{p}\bar{s}$	0.2	1	0.2

Table 3.1: Tolerance degrees with respect to P

In this work, we focus on the original framework [25] where the Pareto method is used to order outcomes.

The Language.

Definition 21. Let A be a finite set of atoms. An ASO theory over A is a tuple (P_{gen}, P_{pref}), where

1. P_{gen}, the generating program, is a logic program, built of atoms in A, used to generate answer sets called feasible outcomes,
2. P_{pref}, the selecting program, is a preference program consisting of preference rules of the form

$$\gamma_1 > \ldots > \gamma_k \leftarrow \alpha,$$

where each γ_i is a propositional formula over A and α is a conjunction of literals of atoms in A.

A single ASO preference rule specifies a total preorder over outcomes. Applying the Pareto method, a general ASO program with multiple ASO rules determines a partial preorder over the space of outcomes represented by answer sets. We will now provide the details.

We say that outcome o is *irrelevant* to preference rule r if $o \models \neg\alpha \vee (\neg\gamma_1 \wedge \ldots \wedge \neg\gamma_k)$, that is, if o does not satisfy α or o does not satisfy any of the propostional formulas γ_i.

As mentioned in the work by Brewka et al [25], outcomes irrelevant to r are considered as good as the best outcomes. This default treatment of irrelevance can be overwritten by including formula $\neg\alpha \vee (\neg\gamma_1 \wedge \ldots \wedge \neg\gamma_k)$ in any place of the preference rule r. Formally we define satisfaction degree of an answer set with respect to a preference rule.

Definition 22. Let o be an outcome generated by P_{gen}, r an ASO preference rule. The satisfaction degree of o on r, denoted $d_r(o)$, is defined as follows: $d_r(o) = 1$ if o is irrelevant to r; $d_r(o) = \min\{i : o \models \gamma_i\}$, otherwise.

Definition 23. Let (P_{gen}, P_{pref}) be an ASO theory, o and o' two outcomes. Outcome o' is weakly Pareto-preferred to o, $o' \succeq o$, if, for every rule r in $P_{pref}, d_r(o') \leq d_r(o)$. outcome o' is strictly Pareto-preferred to o, $o' \succ o$, if $o' \succeq o$ and $d_r(o') < d_r(o)$ for some $r \in P_{pref}$. Outcome o is optimal if there exists no outcome o'' such that $o'' \succ o$.

Consider an ASO theory $P = (P_{gen}, P_{pref})$ over the domain of cars, where

$$P_{gen} = \{\ 1\{c, \bar{c}\}1.1\{p, \bar{p}\}1.1\{s, \bar{s}\}1. \text{ and}$$
$$P_{pref} = \{r.\ \ r'.\},$$

where rule r is $s > \bar{s} \leftarrow c \wedge p$, and rule r' is $\bar{p} \wedge s > p \wedge s > \bar{s}$. Rule r expresses that, among high-capacity and high-price cars, cars of high safety are preferred to cars of low safety. Rule r' describes the preference statement that high-safety and low-price cars are the most preferred, followed by high-safety and high-price cars, which are better than low-safety cars. We now compute the satisfaction degrees of outcomes with regard to P and show the results in Table 3.2. The theory P, thus, induces a total preorder:

$$c\bar{p}s \approx \bar{c}\bar{p}s \succ cps \approx \bar{c}ps \succ c\bar{p}\bar{s} \approx \bar{c}p\bar{s} \approx \bar{c}\bar{p}\bar{s} \succ cp\bar{s}.$$

In this case, we have two optimal outcomes: $c\bar{p}s$ and $\bar{c}\bar{p}s$.

Outcomes	$d_r(o)$	$d_{r'}(o)$
cps	1	2
$cp\bar{s}$	2	3
$c\bar{p}s$	1	1
$c\bar{p}\bar{s}$	1	3
$\bar{c}ps$	1	2
$\bar{c}p\bar{s}$	1	3
$\bar{c}\bar{p}s$	1	1
$\bar{c}\bar{p}\bar{s}$	1	3

Table 3.2: Satisfaction degrees with respect to P

Problems and Complexity. Brewka et al [25] proved that the ASO-DOMINANCE problem is in P, ASO-OPTIMALITY-I is NP-complete, ASO-OPTIMALITY-II is coNP-complete, and ASO-OPTIMALITY-III is Σ_2^P-complete. More recently, Zhu and Truszczynski [79] presented complexity results concerning the existence of optimal outcomes similar and dissimilar to a given interpretation.

Brewka et al [25] also introduced a ranked version of ASO. Ranked ASO programs are ASO programs where rules in P_{pref} are given numeric values that represent different levels of importance of preference rules. Let us assume $P_{pref} = \{P_1, \ldots, P_g\}$ is a collection of ranked ASO preferences divided into g sets P_i, with each set P_i consisting of ASO-rules of rank d_i so that $d_1 < d_2 < \ldots d_g$. We assume that a lower rank of a preference rule indicates its higher importance. We define $o' \succeq^{rk} o$ w.r.t P if for every i, $1 \leq i \leq g$, $o' \approx_{P_i} o$, or if there exists a rank i such that $o' \approx_{P_j} o$ for every j, $j < i$, and $o' \succ_{P_i} o$.

Complexity results discussed above stay unchanged, when we move from unranked to ranked ASO programs.

3.2 Social Choice

The study of preference aggregation can be traced back to social choice theory, which dates back to Condorcet's paradox of voting, noted by the Marquis de Condorcet in the 18th century, in which the winning ranking of outcomes could be cyclic even given

acyclic individual votes [75]. Kenneth Arrow's work, *Social Choice and Individual Values*, is recognized as the basis of modern social choice [1]. In the book, Arrow states that any preference aggregation method for at least three outcomes cannot meet some fairly desirable axioms, a result known as the Arrow's impossibility theorem. Further extending this result, Gibbard and Satterthwaite showed that any social choice function, again meeting some fair properties, is subject to manipulation [43, 69]. Extending the Gibbard-Satterthwaite theorem, the Duggan-Schwartz theorem deals with voting rules that elect a nonempty set of co-winners rather than a single winner [31].

All these results inform us that it is impossible to design a fair preference aggregation system that is manipulation-proof. However, Bartholdi, Tovey and Trick proposed the idea of protecting social choice schemes from manipulation via computational complexity [12, 11, 13]. The idea is that, if manipulation is computationally hard to achieve, manipulation is unlikely.

That started the field of computational social choice by adding an algorithmic perspective from computer science to the formal approach of social choice theory [20].

Preference Aggregation and Voting Rules

One of the most fundamental problems in social choice theory is how to aggregate individual preferences over outcomes so that a collaborative preference relation is reached. In other settings, people are interested in some optimal outcomes rather than a collective preference relation over all outcomes.

Social Welfare and Social Choice Functions.

Definition 24. Let $A = \{a_1, \ldots, a_m\}$ be a finite set of outcomes, $N = \{1, \ldots, n\}$ a finite set of agents (or voters). A preference relation (or a vote) v_i given by agent i is

a total order $\succ_i$, that is, a total, transitive and antisymmetric. A preference profile P is a finite set of preference relations $\{\succ_1, \ldots, \succ_n\}$.

We denote by $\mathcal{L}(A)$ the set of all preference relations over the space of outcomes A, and $\mathcal{L}(A)^n$, the set of all preference profiles.

Definition 25. A social welfare function (SWF) is a function f:

$$\mathcal{L}(A)^n \to \mathcal{L}(A).$$

We call the resulting relation $\succ \in \mathcal{L}(A)$ the social preference relation.

If there are two outcomes a_1 and a_2, May's theorem [63] suggests that a_1 should be preferred to a_2 in the social preference relation if and only if more agents prefer a_1 to a_2 than a_2 to a_1. This idea is called the majority voting. However, when there are more than two outcomes, the majority voting rule can lead to cycles of outcomes, which is known as the Condorcet's paradox. For instance, we have three voters with the following preference relations:

$$a_1 \succ_1 a_2 \succ_1 a_3$$

$$a_2 \succ_2 a_3 \succ_2 a_1$$

$$a_3 \succ_3 a_1 \succ_3 a_2$$

Based on the pairwise majority rule, we have the following cycle

$$a_1 \succ a_2,\ a_2 \succ a_3,\ a_3 \succ a_1.$$

Definition 26. A social choice function (SCF) is a function f:

$$\mathcal{L}(A)^n \to 2^A - \{\emptyset\}.$$

We call the resulting outcome (outcomes) a winner (co-winners, respectively).

Voting Rules

The problem of aggregating individual preferences (or votes) into a single collective preference relation or a single group preferred winner is one of the key problems in social choice theory. Several voting rules and schemas have been proposed over the years. While, when there are three or more candidates, none of these methods is free of some unexpected properties, some of them have gained broad acceptance. I will now introduce some of these commonly used voting rules.

Definition 27. A voting rule r is a specific *SCF* proposed for practical use.

Positional Scoring Rules. For profiles over a set A of outcomes, a *scoring vector* is a sequence $w = (w_1, \ldots, w_m)$ of integers such that $w_1 \geq w_2 \geq \ldots \geq w_m$ and $w_1 > w_m$. Given a vote v with the outcome a in position i ($1 \leq i \leq m$), the score of a in v is given by $s_w(v, a) = w_i$. Given a profile P of votes and an outcome a, the score of a in P is given by $s_w(P, a) = \sum_{v \in P} s_w(v, a)$. These scores determine the ranking generated from P by the scoring vector w (assuming, as is common, some independent tie breaking rule). Common positional scoring rules include the plurality rule, the veto rule, the k-approval rule and Borda's rule.

1. plurality: $(1, 0, \ldots, 0)$
2. veto: $(1, \ldots, 1, 0)$
3. k-approval: $(1, \ldots 1, 0, \ldots 0)$ with k the number of 1's
4. Borda: $(m-1, m-2, \ldots, 1, 0)$

We propose yet another positional scoring rule, called (k, l)-approval [57], with the scoring vector $(a, \ldots, a, b, \ldots, b, 0, \ldots, 0)$, where both a and b are constants ($a \geq b$) and the numbers of a's and b's equal to k and l, respectively. Note that (k, l)-approval allows agents to specify two levels of approval, compared to only one level in k-approval, and thus (k, l)-approval generalizes k-approval.

A voting method, that is closely related to positional scoring rules, is the approval voting [19]. Under approval voting, each voter approves any number of outcomes and the winner, or co-winners, are those with the highest score.

Condorcet Consistent Rules. A *Condorcet winner* is an outcome that wins every pairwise comparisons against each of the other outcomes. Clearly, a Condorcet winner is unique whenever it exists. If a voting rule r always selects the Condorcet winner, if it exists, then r is said to be Condorcet consistent.

Positional scoring rules are not Condorcet consistent [33]. Voting rules that are Condorcet consistent include the following, only to list a few [20].

1. Copeland's rule: An outcome scores 1 for each pairwise comparison it wins, and some number between 0 and 1 for each pairwise comparison it ties. Alternatives with the highest score are the co-winners.
2. Maximin: The Maximin score of an outcome a is the minimum number of votes for a among all pairwise comparisons. Alternatives with the highest Maximin score wins.
3. Kemeny's rule: It selects linear rankings that maximize the number of agreements with pairwise preferences of outcomes in the profile of votes, and the top-ranked outcomes in these rankings are the co-winners.
4. Dodgson's rule: A winner is an outcome that can be made a Condorcet winner by a minimal number of swaps of adjacent outcomes in the votes.

If it is required that only a single winner is eventually elected, we apply some tie-breaking method in case of co-winners. Such a tie-breaking method could be that we break ties in favor of the lexicographically smallest or largest outcome, or in favor of a randomly picked outcome among co-winners.

Manipulation

In practice, preference relations, or votes, are collected from voters who could report preferences that are not truthful. The reason why any agent would do so is that casting an untruthful vote, under certain circumstances, may benefit the agent in that she could end up with a better result, compared to the result of the election with her true preferences. In cases when an agent can be better off misreporting her true preferences, we say that she can perform manipulation. Assuming a voting rule is resolute, we define manipulability as follows.

Definition 28. A resolute voting rule f is manipulable if there exist a voter $i \in N$ and preference profiles $P, P' \in \mathcal{L}(A)^n$ such that $P - v_i = P' - v_i'$ and $f(P) \succ_i f(P')$. A voting rule is strategy-proof if not manipulable.

Since manipulation seems undesirable, researchers are interested in determining what voting rules are manipulable and what are not. Surprisingly, as shown in the Gibbard-Satterthwaite theorem [43, 69], any reasonable voting rule is susceptible to manipulation given that the number of outcomes is at least three.

Theorem 1. (Gibbard, 1973; Satterthwaite, 1975). *When there are at least three outcomes, every resolute, surjective, strategy-proof voting rule is dictatorial.*

Since it is impossible to have any reasonable voting rule that is strategy-proof, researchers have worked hard to circumvent the theorem for strategy-proof voting rules by restricting the domains of preferences[20]. Moulin [64] observed that any voting rule uniquely selecting the Condorcet winner is strategy-proof, assuming the existence of Condorcet winners in any profile of preference relations expressed in a restricted domain. One such domain is the domain of single-peaked preferences [73].

Computational Hardness of Manipulation. In spite of the fact that voting rules can be strategy-proof when preferences are restricted, in practice we cannot impose

constraints onto how agents should formulate their preferences. Another approach to work around the impossibility theorem is to protect voting rules against manipulation by showing high computational complexity.

Chapter 4 Reasoning with Preference Trees

Preference trees, or *P-trees* for short, offer an intuitive and often concise way of representing preferences over combinatorial domains. In this paper, we propose an alternative definition of P-trees, and formally define their compact representation that exploits occurrences of identical subtrees. We show that P-trees generalize lexicographic preference trees and are strictly more expressive. We relate P-trees to *answer-set optimization* programs and *possibilistic logic* theories. Finally, we study reasoning with P-trees and establish computational complexity results for key reasoning tasks of comparing outcomes with respect to orders defined by P-trees, and of finding optimal outcomes.

4.1 Introduction

Let us consider preferences on possible ways to arrange a vacation. We will assume that vacations are described by four binary variables:

1. *activity* (X_1) with values *water sports* (x_1) and *hiking* ($\neg x_1$),
2. *destination* (X_2) with *Florida* (x_2) and *Colorado* ($\neg x_2$),
3. *time* (X_3) with *summer* (x_3) and *winter* ($\neg x_3$), and
4. the mode of *travel* (X_4) could be *car* (x_4) and *plane* ($\neg x_4$).

A complete and consistent set of literals $\neg x_1 \neg x_2 x_3 x_4$ represents the hiking vacation in Colorado in the summer to which we travel by car.

To describe sets of vacations we can use formulas. For instance, vacations that take place in the summer (x_3) or involve water sports (x_1) can be described by the

formula $x_3 \vee x_1$, and vacations in Colorado ($\neg x_2$) that we travel to by car (X_4) by the formula $\neg x_2 \wedge x_4$.

Explicitly specifying strict preference orders on $CD(\mathcal{I})$ becomes impractical even for combinatorial domains with as few as 7 or 8 attributes. However, the setting introduced above allows us to specify total preorders on outcomes in terms of desirable properties outcomes should have. For instance, a formula φ might be interpreted as a definition of a total preorder in which outcomes satisfying φ are preferred to those that do not satisfy φ (and outcomes within each of these two groups are equivalent). More generally, we could see an expression (a sequence of formulas)

$$\varphi_1 > \varphi_2 > \ldots > \varphi_k$$

as a definition of a total preorder in which outcomes satisfying φ_1 are preferred to all others, among which outcomes satisfying φ_2 are preferred to all others, etc., and where outcomes not satisfying any of the formulas φ_i are least preferred. This way of specifying preferences is used (with minor modifications) in possibilistic logic [30] and ASO programs [24]. In our example, the expression

$$x_3 \wedge x_4 > \neg x_3 \wedge \neg x_2$$

states that we prefer summer vacations (x_3) where we drive by car (x_4) to vacations in winter ($\neg x_3$) in Colorado ($\neg x_2$), with all other vacations being the least preferred.

This linear specification of preferred formulas is sometimes too restrictive. An agent might prefer outcomes that satisfy a property φ to those that do not. Within the first group that agent might prefer outcomes satisfying a property ψ_1 and within the other a property ψ_2. Such *conditional* preference can be naturally captured by a form of a decision tree presented in Figure 4.1. Leaves, shown as boxes, represent sets of outcomes satisfying the corresponding conjunctions of formulas ($\varphi \wedge \psi_1$, $\varphi \wedge \neg\psi_1$,

etc.).

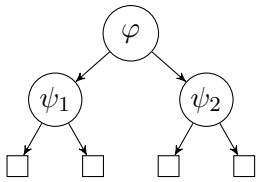


Figure 4.1: A preference tree

Trees such as the one in Figure 4.1 are called *preference trees*, or *P-trees*. They were introduced by Fraser [35, 36], who saw them as a convenient way to represent conditional preferences. Despite their intuitive nature they have not attracted much interest in the preference research in AI. In particular, they were not studied for their relationship to other preference formalisms. The attribute of compact representations received only an informal treatment by Fraser (P-trees in their full representation are often impractically large), and the algorithmic attributes of reasoning with P-trees were also only touched upon.

We propose an alternative definition of preference trees, and formally define their compact representation that exploits occurrences of identical subtrees. P-trees are reminiscent of LP-trees [16]. We discuss the relation between the two concepts and show that P-trees offer a much more general, flexible and expressive way of representing preferences. We also discuss the relationship between preference trees and ASO preferences and possibilistic logic theories. We study the complexity of problems of comparing outcomes with respect to orders defined by preference trees, and of problems of finding optimal outcomes.

This chapter is organized as follows. In the next section, we formally define P-trees and a compact way to represent them. In the following section we present results comparing the language of P-trees with other preference formalisms. We then move on to study the complexity of key reasoning tasks for preferences captured by P-trees and, finally, conclude by outlining some future research directions.

4.2 Preference Trees

In this section, we define preference trees and discuss their representation. Let $\mathcal{I}$ be a set of binary attributes[1]. A *preference tree* (*P-tree*, for short) over $\mathcal{I}$ is a binary tree with all nodes other than leaves labeled with propositional formulas over $\mathcal{I}$. Each P-tree T defines a natural strict order $\succeq_T$ on the set of its leaves, the order of their enumeration from left to right.

Given an outcome $o \in CD(\mathcal{I})$, we define the *leaf of o in T* as the leaf reached by starting at the root of T and proceeding downwards. When at a node t labeled with φ, if $o \models \varphi$, we descend to the left child of t; otherwise, we descend to the right node of t. We denote the leaf of o in T by $l_T(o)$.

We use the concept of the leaf of an outcome o in a P-tree T to define a total preorder on $CD(\mathcal{I})$. Namely, for outcomes $o_1, o_2 \in CD(\mathcal{I})$, we set $o_1 \succeq_T o_2$, o_1 is *preferred* to o_2, if $l_T(o_1) \succeq_T l_T(o_2)$, and $o_1 \succ_T o_2$, o_1 is *strictly preferred* to o_2, if $l_T(o_1) \succ_T l_T(o_2)$. (We overload the relations $\succeq_T$ and $\succ_T$ by using it both for the order on the leaves of T and the corresponding preorder on the outcomes from $CD(\mathcal{I})$). We say that o_1 is *equivalent* to o_2, $o_1 \approx_T o_2$, if $l_T(o_1) = l_T(o_2)$. Finally, o is *optimal* if there exists no o' such that $o' \succ_T o$.

Let us come back to the vacation example and assume that an agent prefers vacations involving water sports in Florida or hiking in Colorado over the other options. This preference is described by the formula $(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$ or, more concisely, as an equivalence $x_1 \equiv x_2$. Within each of the two groups of vacations (satisfying the formula and not satisfying the formula), driving (x_4) is the preferred transporting mode. These preferences can be captured by the P-tree in Figure 4.2a. We note that in this example, the preferences at the second level are *unconditional*, that is, they do not depend on preferences at the top level.

To compare two outcomes, $o_1 = \neg x_1 \neg x_2 \neg x_3 x_4$ and $o_2 = x_1 x_2 x_3 \neg x_4$, we walk down

[1] In case of multi-value attributes, $\mathcal{I}$ is then a set of binary variables representing attribute values.

(a) Full (b) Compact

Figure 4.2: P-trees on vacations

the tree and find that $l_T(o_1) = l_1$ and $l_T(o_2) = l_2$. Thus, we have $o_1 \succ_T o_2$ since l_1 precedes l_2.

The key property of P-trees is that they can represent any total preorder on $CD(\mathcal{I})$.

Proposition 1. *For every set $\mathcal{I}$ of binary attributes, for every set $D \subseteq CD(\mathcal{I})$ of outcomes over $\mathcal{I}$, and for every total preorder $\succeq$ on D into no more than 2^n clusters of equivalent outcomes, there is a P-tree T of depth at most n such that the preorder determined by T on $CD(\mathcal{I})$ when restricted to D coincides with $\succeq$ (that is, $\succeq_{T|D} = \succeq$).*

Proof. Let $\succeq$ be a total preorder on a subset $D \subseteq CD(\mathcal{I})$ of outcomes over $\mathcal{I}$, and let $D_1 \succ D_2 \succ \ldots \succ D_m$ be the corresponding strict ordering of clusters of equivalent outcomes, with $m \leq 2^n$. If $m = 1$, a single-leaf tree (no decision nodes, just a box node) represents this preorder. This tree has depth 0 and so, the assertion holds. Let us assume then that $m > 1$, and let us define $D' = D_1 \cup \ldots \cup D_{\lceil m/2 \rceil}$ and $D'' = D \setminus D'$. Let $\varphi_{D'}$ be a formula such that models of D' are precisely the outcomes in D' (such a formula can be constructed as a disjunction of conjunctions of literals, each conjunction representing a single outcome in D'). If we place $\varphi_{D'}$ in the root of a P-tree, that tree represents the preorder with two clusters, D' and D'', with D' preceding D''. Since each of D' and D'' has no more than 2^{n-1} clusters, by induction, the preorders $D_1 \succ \ldots \succ D_{\lceil m/2 \rceil}$ and $D_{\lceil m/2 \rceil + 1} \succ \ldots \succ D_m$ can each be represented as a P-tree with depth at most $n - 1$. Placing these trees as the left and the right

subtrees of $\varphi_{D'}$ respectively results in a P-tree of depth at most n that represents $\succeq$. □

Compact Representation of P-Trees. Proposition 2 shows high expressivity of P-trees. However, the construction described in the proof has little practical use. First, the P-tree it produces may have a large size due to the large sizes of labeling formulas that are generated. Second, to apply it, one would need to have an explicit enumeration of the preorder to be modeled, and that explicit representation in practical settings is unavailable.

However, preferences over combinatorial domains that arise in practice typically have structure that can be elicited from a user and exploited when constructing a P-tree representation of the preferences. First, decisions at each level are often based on considerations involving only very few attributes, often just one or two and very rarely more than that. Moreover, the subtrees of a node that order the "left" and the"right" outcomes are often identical or similar.

Exploiting these features often leads to much smaller representations. A *compact P-tree over* $\mathcal{I}$ is a tree such that

1. every node is labeled with a Boolean formula over $\mathcal{I}$, and
2. every non-leaf node t labeled with φ has either two outgoing edges, with the left one meant to be taken by outcomes that satisfy φ and the right one by those that make φ false (Figure 4.3a), or one outgoing edge pointing
   - straight-down (Figure 4.3b), which indicates that the two subtrees of t are *identical* and the formulas labeling every pair of corresponding nodes in the two subtrees are the *same*,
   - left (Figure 4.3c), which indicates that right subtree of t is empty, or
   - right (Figure 4.3d), which indicates that left subtree of t is empty.

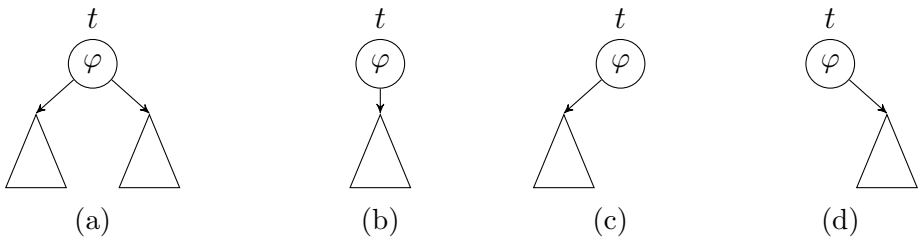
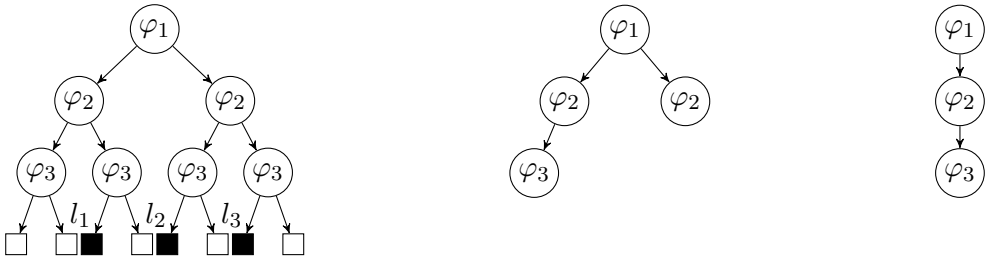


Figure 4.3: Compact P-trees

The P-tree in Figure 4.2a can be collapsed as both subtrees of the root are the same (including the labeling formulas). This leads to a tree in Figure 4.2b with a straight-down edge. We note that we drop box-labeled leaves in compact representations of P-trees, as they no longer have an interpretation as distinct clusters.

Empty Leaves in P-Trees. Given a P-tree T one can prune it so that all sets of outcomes corresponding to its leaves are non-empty. However, keeping empty clusters may lead to compact representations of much smaller (in general, even exponentially smaller) size.

A full P-tree T in Figure 4.4a uses labels $\varphi_1 = \neg x_1 \vee x_3$, $\varphi_2 = x_2 \vee \neg x_4$, and $\varphi_3 = x_2 \wedge x_3$. We check that leaves l_1, l_2 and l_3 are empty, that is, the conjunctions $\varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3$, $\neg\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ and $\neg\varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3$ are unsatisfiable. Pruning T one obtains a compact tree T' (Figure 4.4b) that is smaller compared to T, but larger than T'' (Figure 4.4c), another compact representation of T, should we allow empty leaves and exploit the structure of T.



(a) T (b) T': pruned T (c) T''

Figure 4.4: P-trees with empty leaves

That example generalizes and leads to the question of finding small sized representations of P-trees. (We conjecture that the problem in its decision version asking about the existence of a compact representation of size at most k is NP-complete). From now on, we assume that P-trees are given in their compact representation.

4.3 P-Trees and Other Formalisms

In this section we compare the preference representation language of P-trees with other preference languages.

P-Trees Generalize LP-Trees. As stated earlier, P-trees are reminiscent of LP-trees, a preference language that has received significant attention recently [16, 53, 57]. In fact, LP-trees over a set $\mathcal{I} = \{x_1, \ldots, x_n\}$ of attributes are simply special P-trees over $\mathcal{I}$. Namely, an LP-tree over $\mathcal{I}$ can be defined as a P-tree over $\mathcal{I}$, in which all formulas labeling nodes are atoms x_i or their negations $\neg x_i$, depending on whether x_i or $\neg x_i$ is the preferred over the other, and every path from the root to a leaf has all atoms x_i appear in it as labels exactly once. Clearly, LP-trees are full binary trees of depth n (assuming the depth of the root is 1) and determine strict *total orders* on outcomes in $CD(\mathcal{I})$ (no indifference between different outcomes). An example of an LP-tree over $\{x_1, x_2, x_3, x_4\}$ for our vacation example is given in Figure 4.5.

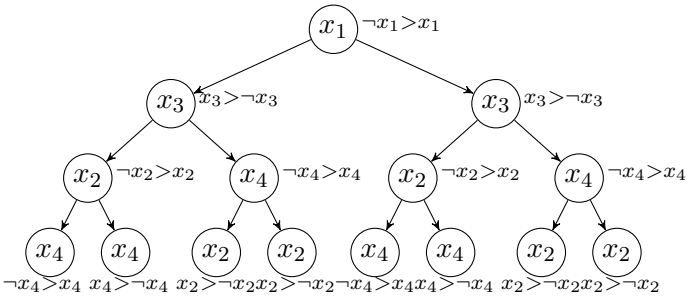
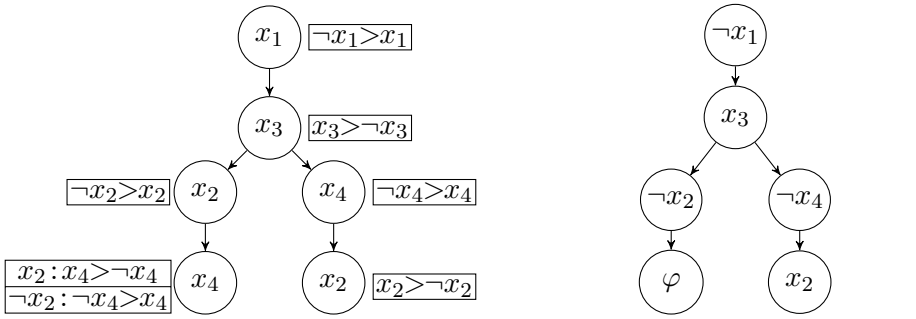


Figure 4.5: A full LP-tree on vacations

In general representing preferences by LP-trees is impractical. The size of the representation is of the same order as that of an explicit enumeration of the preference

order. However, in many cases preferences on outcomes have structure that leads to LP-trees with similar subtrees. That structure can be exploited, as in P-trees, to represent LP-trees compactly. Figure 4.6a shows a compact representation of the LP-tree in Figure 4.5. We note the presence of conditional preference tables that make up for the lost full binary tree structure. Together with the simplicity of the language, compact representations are behind the practical usefulness of LP-trees. The compact representations of LP-trees translate into compact representations of P-trees, in the sense defined above. This matter is not central to our discussion and we simply illustrate it with an example. The compactly represented P-tree in Figure 4.6b is the counterpart to the compact LP-tree in Figure 4.6a, where $\varphi = (x_2 \wedge x_4) \vee (\neg x_2 \wedge \neg x_4)$.



(a) A compact LP-tree (b) The corresponding P-tree

Figure 4.6: A compact LP-tree as a compact P-tree

The major drawback of LP-trees is that they can capture only a very small fraction of preference orders. One can show that the number, say $G(n)$, of LP-trees over n attributes is

$$G(n) = \prod_{k=0}^{n-1} (n-k)^{2^k} \cdot 2^{2^k}$$

and is asymptotically much smaller than $L(n) = (2^n)!$, the number of all preference orders of the corresponding domain of outcomes. In fact, one can show that

$$\frac{G(n)}{L(n)} < \frac{1}{2^{(2^n \cdot (n - \log n - 2))}}.$$

This is in stark contrast with Proposition 2, according to which every total preorder can be represented by a P-tree.

Even very natural orderings, which have simple (and compact) representations by P-trees often cannot be represented as LP-trees. For instance, there is no LP-tree on $\{x_1, x_2\}$ representing the order $00 \succ 11 \succ 01 \succ 10\}$. However, the P-trees (both full and compact) in Figure 4.2 do specify it.

P-Trees Extend ASO-Rules. The formalism of ASO-rules [24] provides an intuitive way to express preferences over outcomes as total preorders. An ASO-rule partitions outcomes into ordered clusters according to the semantics of the formalism. Formally, an ASO-rule r over $\mathcal{I}$ is a preference rule of the form

$$C_1 > \ldots > C_m \leftarrow B, \tag{4.1}$$

where all C_i's and B are propositional formulas over $\mathcal{I}$. For each outcome M, rule (4.1) determines its *satisfaction degree*. It is denoted by $SD_r(M)$ and defined by

$$SD_r(M) = \begin{cases} 1, & M \models \neg B \\ m+1, & M \models B \wedge \bigwedge_{1 \leq i \leq m} \neg C_i \\ min\{i : M \models C_i\}, & \text{otherwise.} \end{cases}$$

We say that an outcome M is weakly preferred to an outcome M' ($M \succeq_r M'$) if $SD_r(M) \leq SD_r(M')$. Thus, the notion of the satisfaction degree (or, equivalently, the preference r) partitions outcomes into (in general) $m+1$ clusters.[2]

Let us consider the domain of vacations. An agent may prefer hiking in Colorado to water sports in Florida if she is going on a summer vacation. Such preference can be described as an ASO-rule:

$$\neg x_1 \wedge \neg x_2 > x_1 \wedge x_2 \leftarrow x_3.$$

[2]This definition is a slight adaptation of the original one.

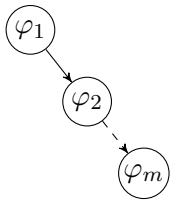


Figure 4.7: A P-tree T_r (T_P)

Under the semantics of ASO, this preference rule specifies that the most desirable vacations are summer hiking vacations to Colorado and all winter vacations, the next preferred vacations are summer water sports vacations to Florida, and the least desirable vacations are summer hiking vacations to Florida and summer water sports vacations to Colorado.

Given an ASO-rule r of form (4.1), we show how r is encoded in a P-tree. From the ASO-rule r, we build a P-tree T_r in Figure 4.7, where $\varphi_1 = \neg B \vee C_1$, $\varphi_i = C_i$ ($2 \leq i \leq m$), and the dashed edge represents nodes labeled by the formulas $\varphi_3, \ldots, \varphi_{m-1}$ and every formula φ_i, $3 \leq i \leq m-1$, is constructed such that the parent of φ_i is φ_{i-1}, the left child of φ_i is empty, and the right child of φ_i is φ_{i+1}.

Theorem 2. *Given an ASO-rule r, the P-tree T_r has size linear in the size of r, and for every two outcomes M and M'*

$$M \succeq_r^{ASO} M' \text{ iff } M \succeq_{T_r} M'$$

Proof. The P-tree T_r induces a total preorder $\succeq_{T_r}$ where outcomes satisfying φ_1 are preferred to outcomes satisfying $\neg\varphi_1 \wedge \varphi_2$, which are then preferred to outcomes satisfying $\neg\varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3$, and so on. The least preferred are the ones satisfying $\bigwedge_{1 \leq i \leq m} \neg\varphi_i$. Clearly, this order $\succeq_{T_r}$ is precisely the order $\succeq_r^{ASO}$ given by the ASO rule r. □

There are other ways of translating ASO-rules to P-trees. For instance, it might be beneficial if the translation produced a more balanced tree. Keeping the definitions

of φ_i, $1 \leq i \leq m$, as before and setting $\varphi_{m+1} = B \wedge \neg C_1 \wedge \ldots \wedge \neg C_m$, we could proceed as in the proof of Proposition 2.

For example, if $m = 6$, we build the P-tree T_r^b in Figure 4.8, where $\psi_1 = \varphi_1 \vee \varphi_2 \vee \varphi_3 \vee \varphi_4$, $\psi_2 = \varphi_1 \vee \varphi_2$, $\psi_3 = \varphi_1$, $\psi_4 = \varphi_3$, $\psi_5 = \varphi_5 \vee \varphi_6$, and $\psi_6 = \varphi_5$. The indices i's of the formulas ψ_i's indicate the order in which the corresponding formulas are built recursively.

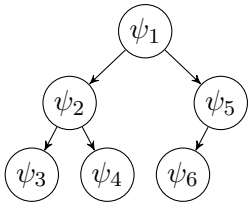


Figure 4.8: T_r^b when $m = 6$

This P-tree representation of a preference r of the form (4.1) is balanced with height $\lceil \log_2(m+1) \rceil$. Moreover, the property in Theorem 2 also holds for the balanced T_r^b of size polynomial in the size of r. In fact, the size of T_r^b is in $O(s_r \log s_r)$, where s_r is the size of rule r. It is clear that, though tree T_r^b is larger than T_r in size, comparing outcomes could be done faster due to a smaller depth of T_r^b.

Representing P-Trees as RASO-Theories. Preferences represented by compact P-trees cannot in general be captured by ASO preferences without a significant (in some cases, exponential) growth in the size of the representation. However, any P-tree can be represented as a set of *ranked* ASO-rules, or an RASO-theory [24], aggregated by the Pareto method.

Given a P-tree T, we construct an RASO-theory Φ_T as follows. We start with $\Phi_T = \emptyset$. For every node t_i in a P-tree T, we update $\Phi_T = \Phi_T \cup \{\varphi_i \overset{d_i}{\leftarrow} \mathit{conditions}\}$, where φ_i is the formula labeling node t_i, d_i, rank of the ASO-rule, is the depth of node t_i, and *conditions* is the conjunction of formulas φ_j or $\neg\varphi_j$ labeling all nodes t_j that are ancestor nodes of t_i in T with two outgoing edges. Whether φ_j or $\neg\varphi_j$

is used depends on how the path from the root to t_i determines whether descending left (φ_j) or right ($\neg\varphi_j$) at t_j.

For instance, the P-tree T in Figure 4.6b gives rise to the following RASO-theory:

$\neg x_1 \overset{1}{\leftarrow}.$

$x_3 \overset{2}{\leftarrow}.$

$\neg x_2 \overset{3}{\leftarrow} x_3. \qquad \neg x_4 \overset{3}{\leftarrow} \neg x_3.$

$(x_2 \wedge x_4) \vee (\neg x_2 \wedge \neg x_4) \overset{4}{\leftarrow} x_3. \qquad x_2 \overset{4}{\leftarrow} \neg x_3.$

Theorem 3. *Given a P-tree T, there exists an RASO-theory Φ_T of size polynomial in the size of T such that for every two outcomes M and M'*

$$M \succeq_{\Phi_T}^{RASO} M' \text{ iff } M \succeq_T M'$$

Proof. ($\Leftarrow$) Let us assume $M \succeq_T M'$. Denote by $(\varphi_{i_1}, \ldots, \varphi_{i_j})$ the order of formulas labeling the path determined by M from the root to a leaf. Let φ_{i_k}, $1 \leq k \leq j$, be the first formula that M and M' evaluate differently, in fact, $M \models \varphi_{i_k}$ and $M' \not\models \varphi_{i_k}$. Denote by d the depth of φ_{i_k} in T. Based on the construction of Φ_T, for every RASO-rule r of rank less than d, we have $M \approx_r^{ASO} M'$. For every RASO-rule r of rank d, we have $M \succ_r^{ASO} M'$ if r comes from φ_{i_k}; $M \approx_r^{ASO} M'$ for other rules of rank d. According to RASO ordering, $M \approx_{\Phi_T}^{RASO} M'$ holds if φ_{i_k} does not exist; $M \succ_{\Phi_T}^{RASO} M'$ holds, otherwise. Therefore, $M \succeq_{\Phi_T}^{RASO} M'$ holds.

($\Rightarrow$) Prove by contradiction. We assume that $M \succeq_{\Phi_T}^{RASO} M'$ and $M' \succ_T M$ hold. We again denote by $(\varphi_{i_1}, \ldots, \varphi_{i_j})$ the order of formulas labeling the path determined by M from the root to a leaf. There must exist some formula φ_{i_k}, $1 \leq k \leq j$, such that $M' \models \varphi_{i_k}$, $M \not\models \varphi_{i_k}$, and all formulas φ_ℓ, $1 \leq \ell \leq k-1$, are evaluated in the same way by M and M'. Based on RASO ordering, we have $M' \succ_{\Phi_T}^{RASO} M$, contradiction. □

Hence, the relationship between P-trees and ASO preferences can be summarized as follows. Every ASO preference rule can be translated into a P-tree, and every P-tree into a theory of ranked ASO preference rules. In both cases, the translations have size polynomial in the size of the input. Examining the reverse direction, the size of the ASO rule translated from a P-tree could be exponential, and the orders represented by ranked ASO theories *strictly include* the orders induced by P-trees as RASO-theories describe *partial* preorders in general.

P-Trees Extend Possibilistic Logic. Similarly as for ASO-rules, we can apply different methods to encode a possibilistic logic theories in P-trees. Here we discuss one of them. We define T_Π to be an unbalanced P-tree shown in Figure 4.7 with with labels φ_i defined as follows: $\varphi_1 = \bigwedge_{1\leq i\leq m} \phi_i$, $\varphi_2 = \bigwedge_{1\leq i\leq m-1} \phi_i \wedge \neg\phi_m$, $\varphi_3 = \bigwedge_{1\leq i\leq m-2} \phi_i \wedge \neg\phi_{m-1}$, and $\varphi_m = \phi_1 \wedge \neg\phi_2$.

Theorem 4. *Given a possibilistic theory* Π*, there exists a P-tree* T_Π *of size polynomial in the size of* Π *such that for every two outcomes* M *and* M'

$$M \succeq_\Pi^{Poss} M' \text{ iff } M \succeq_{T_\Pi} M'$$

Proof. It is clear that the size of P-tree T_Π is polynomial in the size of Π. Let $mi(M, \Pi)$ denote the maximal index j such that M satisfies all $\phi_1, \ldots, \phi_j$ in Π. (If M falsifies all formulas in Π, we have $mi(M, \Pi) = 0$.) One can show that $M \succeq_\Pi^{Poss} M'$ if and only if $mi(M, \Pi) \geq mi(M', \Pi)$, and $mi(M, \Pi) \geq mi(M', \Pi)$ if and only if $M \succeq_{T_\Pi} M'$. Therefore, the theorem follows. □

4.4 Reasoning Problems and Complexity

In this section, we study decision problems on reasoning about preferences described as P-trees, and provide computational complexity results for the three reasoning problems defined below.

Definition 29. Dominance-testing (DomTest): given a P-tree T and two distinct outcomes M and M', decide whether $M \succeq_T M'$.

Definition 30. Optimality-testing (OptTest): given a P-tree T and an outcome M of T, decide whether M is optimal.

Definition 31. Optimality-with-property (OptProp): given a P-tree T and some property α expressed as a Boolean formula over the vocabulary of T, decide whether there is an optimal outcome M that satisfies α.

Our first result shows that P-trees support efficient dominance testing.

Theorem 5. *The* DomTest *problem can be solved in time linear in the height of the P-tree T.*

Proof. The DomTest problem can be solved by walking down the tree. The preference between M and M' is determined at the first non-leaf node n where M and M' evaluate φ_n differently. If such node does not exist before arriving at a leaf, $M \approx_T M'$. □

An interesting reasoning problem not mentioned above is to decide whether there exists an optimal outcome with respect to the order given by a P-tree. However, this problem is trivial as the answer simply depends on whether there is any outcome at all. However, optimality *testing* is a different matter. Namely, we have the following result.

Theorem 6. *The* OptTest *problem is coNP-complete.*

Proof. We show that the complementary problem, testing non-optimality of an outcome M, is NP-complete. Membership is obvious. A witness of non-optimality of M is any outcome M' such that $M' \succ_T M$, a property that can be verified in linear time (cf. Theorem 5). NP-hardness follows from a polynomial time reduction from SAT

Figure 4.9: The P-tree T_Φ

[39]. Given a CNF formula $\Phi = c_1 \wedge \ldots \wedge c_n$ over a set of variables $V = \{X_1, \ldots, X_m\}$, we construct a P-tree T and an outcome M as follows.

1. We choose $X_1, \ldots, X_m, unsat$ as attributes, where $unsat$ is a new variable;

2. we define the P-tree T_Φ (cf. Figure 4.9) to consist of a single node labeled by $\Psi = \Phi \wedge \neg unsat$;

3. we set $M = \{unsat\}$.

We show that $M = \{unsat\}$ is not an optimal outcome if and only if $\Phi = \{c_1, \ldots, c_n\}$ is satisfiable.

($\Rightarrow$) Assume that $M = \{unsat\}$ is not an optimal outcome. Since $M \not\models \Psi$, M belongs to the right leaf and there must exist an outcome M' such that $M' \succ M$. This means that $M' \models \Phi \wedge \neg unsat$. Thus, Φ is satisfiable.

($\Leftarrow$) Let M' be a satisfying assignment to Φ over $\{X_1, \ldots, X_m\}$. Since no $c_i \in \Phi$ mentions $unsat$, we can assume $unsat \notin M'$. So $M' \models \Psi$ and M' is optimal. Thus, $M = \{unsat\}$ is not optimal. □

Theorem 7. *The* OptProp *problem is Δ_2^P-complete.*

Proof. (Membership) The problem is in the class Δ_2^P. Let T be a given preference tree. To check whether there is an optimal outcome that satisfies a property α, we start at the root of T and move down. As we do so, we maintain the information about the path we took by updating a formula ψ, which initially is set to $\top$ (a generic tautology). Each time we move down to the left from a node t, we update ψ to $\psi \wedge \varphi_t$, and when we move down to the right, to $\psi \wedge \neg\varphi_t$. To decide whether to move down left or right form a node t, we check if $\varphi_t \wedge \psi$ is satisfiable by making a call to an NP

oracle for deciding satisfiability. If $\varphi_t \wedge \psi$ is satisfiable, we proceed to the left subtree and, otherwise, to the right one. We then update t to be the node we moved to and repeat. When we reach a leaf of the tree (which represents a cluster of outcomes), this cluster is non-empty, consists of all outcomes satisfying ψ and all these outcomes are optimal. Thus, returning YES, if $\psi \wedge \alpha$ is satisfiable and NO, otherwise, correctly decides the problem. Since the number of oracle calls is polynomial in the size of the tree T, the problem is in the class Δ_2^P.

(Hardness) The maximum satisfying assignment (MSA) problem[3] [51] is Δ_2^P-complete. We first show that MSA remains Δ_2^P-hard if we restrict the input to Boolean formulas that are satisfiable and have models other than the all-false model (i.e., $\neg x_1 \ldots \neg x_n$).

Lemma 1. *The MSA problem is Δ_2^P-complete when Φ is satisfiable and has models other than the all-false model.*

Proof. Given a Boolean formula Φ over $\{x_1, \ldots, x_n\}$, we define $\Psi = \Phi \vee (x_0 \wedge \neg x_1 \wedge \ldots \wedge \neg x_n)$ over $\{x_0, x_1, \ldots, x_n\}$. It is clear that Ψ is satisfiable, and has at least one model other than the all-false one. Let M be a lexicographically maximum assignment satisfying Φ and M has $x_n = 1$. Extending M by $x_0 = 1$ yields a lexicographically maximum assignment satisfying Ψ and this assignment satisfies $x_n = 1$. Conversely, if M is a lexicographically maximum assignment satisfying Ψ and $x_n = 1$ holds in M, it follows that $M \models \Phi$. Thus, restricted M to $\{x_1, \ldots, x_n\}$, the assignment is lexicographically maximal satisfying Φ. □

We now show the hardness of the OptProp problem by a reduction from this restricted version of the MSA problem. Let Φ be a satisfiable propositional formula over variables $x_1, \ldots, x_n$ that has at least one model other than the all-false one. We construct an instance of the OptProp problem as follows. We define the P-tree T_Φ

[3]Given a Boolean formula Φ over $\{x_1, \ldots, x_n\}$, the maximum satisfying assignment (MSA) problem is to decide whether $x_n = 1$ in the lexicographically maximum satisfying assignment for Φ. (If Φ is unsatisfiable, the answer is *no*.)

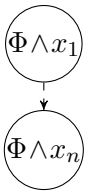


Figure 4.10: The P-tree T_Φ

as shown in Figure 4.10, where every node is labeled by formula $\Phi \wedge x_i$, and we set $\alpha = x_n$.

Our P-tree T_Φ induces a total preorder consisting of a sequence of singleton clusters, each containing an outcome satisfying Φ, followed by a single cluster comprising all outcomes that falsify Φ and the all-false model. By our assumption on Φ, the total preorder has at least two non-empty clusters. Moreover, all singleton clusters preceding the last one are ordered lexicographically. Thus, the optimal outcome of T_Φ satisfies α if and only if the lexicographical maximum satisfying outcome of Φ satisfies x_n. □

4.5 Conclusions

We investigated the qualitative preference representation language of *preference trees*, or *P-trees*. This language was introduced in early 1990s (cf. [35, 36]), but have not received a substantial attention as a formalism for preference representation in AI. We studied formally the attribute of compact representations of P-trees, established its relationship to other preference languages such as lexicographic preference trees, possibilistic logic and answer-set optimization. For several preference reasoning problems on P-trees we derived their their computational complexity.

P-trees are quite closely related to possibilistic logic theories or preference expressions in answer-set optimization. However, they allow for much more structure among formulas appearing in these latter two formalisms (arbitrary trees as opposed to the linear structure of preference formulas in the other two formalisms). This structure

allows for representations of conditional preferences. P-trees are also more expressive than lexicographic preference trees. This is the case even for P-trees in which every node is labeled with a formula involving just two attributes, as we illustrated with the $00 \succ 11 \succ 01 \succ 01$ example. Such P-trees are still simple enough to correspond well to the way humans formulate hierarchical models of preferences, with all their decision conditions typically restricted to one or two attributes.

Our paper shows that P-trees form a rich preference formalism that deserves further studies. Among the open problems of interest are those of learning P-trees and their compact representations, aggregating P-trees coming from different sources (agents), and computing optimal consensus outcomes. These problems will be considered in the future work.

Chapter 5 Learning Partial Lexicographic Preference Trees

We introduce *partial lexicographic preference trees* (PLP-trees) as a formalism for compact representations of preferences over combinatorial domains. Our main results concern the problem of passive learning of PLP-trees. Specifically, for several classes of PLP-trees, we study how to learn (i) a PLP-tree consistent with a dataset of examples, possibly subject to requirements on the size of the tree, and (ii) a PLP-tree correctly ordering as many of the examples as possible in case the dataset of examples is inconsistent. We establish complexity of these problems and, in all cases where the problem is in the class P, propose polynomial time algorithms.

5.1 Introduction

Recently, there has been a rising interest in representing preferences over combinatorial domains by exploiting the notion of the lexicographic ordering. For instance, assuming issues are over the binary domain $\{0, 1\}$, with the preferred value for each issue being 1, a sequence of issues naturally determines an order on outcomes. This idea gave rise to the language of *lexicographic preference models* or *lexicographic strategies*, which has been extensively studied in the literature [70, 28, 78]. The formalism of complete *lexicographic preference trees* (LP-trees) [16] generalizes the language of lexicographic strategies by arranging issues into decision trees that assign preference ranks to outcomes. An important aspect of LP-trees is that they allow us to model *conditional* preferences on issues and *conditional* ordering of issues. Another formalism, the language of *conditional lexicographic preference trees* (or CLP-trees) [21], extends LP-trees by allowing subsets of issues as labels of nodes.

A central problem in preference representation concerns learning implicit models of preferences (such as lexicographic strategies, LP-trees or CLP-trees), of possibly

small sizes, that are consistent with all (or at least possibly many) given examples, each correctly ordering a pair of outcomes. The problem was extensively studied. Booth et al. [16] considered learning of LP-trees, and Bräuning and Eyke [21] of CLP-trees.

In this paper, we introduce *partial lexicographic preference trees* (or *PLP-trees*) as means to represent *total preorders* over combinatorial domains. PLP-trees are closely related to LP-trees requiring that every path in the tree contains all issues used to describe outcomes. Consequently, LP-trees describe total orders over the outcomes. PLP-trees relax this requirement and allow paths on which some issues may be missing. Hence, PLP-trees describe total preorders. This seemingly small difference has a significant impact on some of the learning problems. It allows us to seek PLP-trees that minimize the set of issues on their paths, which may lead to more robust models by disregarding issues that have no or little influence on the true preference (pre)order.

The rest of the paper is organized as follows. In the next section, we introduce the language of PLP-trees and describe a classification of PLP-trees according to their complexity. We also define three types of passive learning problems for the setting of PLP-trees. In the following sections, we present algorithms learning PLP-trees of particular types and computational complexity results on the existence of PLP-trees of different types, given size or accuracy. We close with conclusions and a brief account of future work.

5.2 Partial Lexicographic Preference Trees

Let $\mathcal{I} = \{X_1, \ldots, X_p\}$ be a set of binary issues, with each X_i having its domain $D_i = \{0_i, 1_i\}$. The corresponding *combinatorial domain* is the set $\mathcal{X} = D_1 \times \ldots \times D_p$. Elements of $\mathcal{X}$ are called *outcomes*.

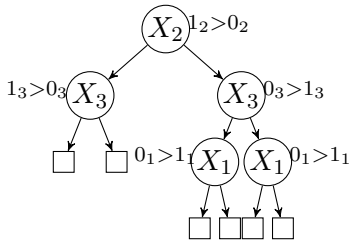
A PLP-tree over $\mathcal{X}$ is binary tree whose every non-leaf node is labeled by an issue

from $\mathcal{I}$ and by a preference entry $1 > 0$ or $0 > 1$, and whose every leaf node is denoted by a box $\Box$. Moreover, we require that on every path from the root to a leaf each issue appears *at most* once.

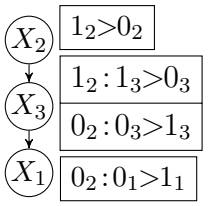
To specify the total preorder on outcomes defined by a PLP-tree T, let us enumerate leaves of T from left to right, assigning them integers $1, 2$, etc. For every outcome α we find its leaf in T by starting at the root of T and proceeding downward. When at a node labeled with an issue X, we descend to the left or to the right child of that node based on the value $\alpha(X)$ of the issue X in α and on the preference assigned to that node. If $\alpha(X)$ is the preferred value, we descend to the left child. We descend to the right child, otherwise. The integer assigned to the leaf that we eventually get to is the *rank* of α in T, written $r_T(\alpha)$. The preorder $\succeq_T$ on distinct outcomes determined by T is defined as follows: $\alpha \succeq_T \beta$ if $r_T(\alpha) \leq r_T(\beta)$ (smaller ranks are "better"). We also define derived relations $\succ_T$ (strict order) and $\approx_T$ (equivalence or indifference): $\alpha \succ_T \beta$ if $\alpha \succeq_T \beta$ and $\beta \not\succeq_T \alpha$, and $\alpha_T \approx_T \beta$ if $\alpha \succeq_T \beta$ and $\beta \succeq_T \alpha$. Clearly, $\succeq_T$ is a total preorder on outcomes partitioning them into strictly ordered clusters of equivalent outcomes.

To illustrate the notions just introduced, we consider preference orderings of dinner options over four binary issues. The *appetizer* (X_1) can be either *salad* (0_1) or *soup* (1_1). The *main course* (X_2) is either *beef* (0_2) or *fish* (1_2). The *drink* (X_3) can be *beer* (0_3) or (white) *wine* (1_3). Finally, *dessert* (X_4) can be *ice-cream* (0_4) or *pie* (1_4). An agent could specify her preferences over dinners as a PLP-tree in Figure 6.2a. The *main course* is the most important issue to the agent and she prefers fish to beef. Her next most important issue is what to *drink* (independently of her selection for the main course). She prefers wine over beer with fish, and beer over wine with beef. If the agent has beef for her main dish, no matter what drink she gets, her next consideration is the *appetizer*, and she prefers salad to soup. In this example, the *dessert* does not figure into preferences at all. The most preferred dinners have fish

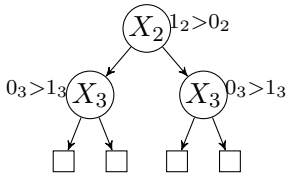
for the main course and wine for the drink, with all possible combinations of choices for the appetizer and dessert (and so, the cluster of most preferred dinners has four elements).



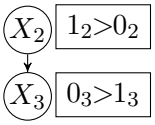
(a) Collapsible PLP-tree



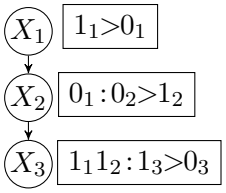
(b) UI-CP PLP-tree



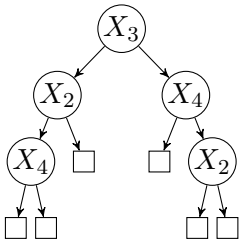
(c) Collapsible PLP-tree



(d) UI-UP PLP-tree



(e) Invalid UI-CP PLP-tree



(f) CI-FP PLP-tree

Figure 5.1: PLP-trees over the dinner domain

Classification of PLP-Trees

In the worst case, the size of a PLP-tree is exponential in the number of issues in $\mathcal{I}$. However, some PLP-trees have a special structure that allows us to "collapse" them and obtain more compact representations. This yields a natural classification of PLP-trees, which we describe below.

Let $R \subseteq \mathcal{I}$ be the set of issues that appear in a PLP-tree T. We say that T is *collapsible* if there is a permutation $\hat{R}$ of elements in R such that for every path in T from the root to a leaf, issues that label nodes on that path appear in the same order in which they appear in $\hat{R}$.

If a PLP-tree T is collapsible, we can represent T by a single path of nodes labeled with issues according to the order in which they occur in $\hat{R}$, where a node labeled with an issue X_i is also assigned a *partial conditional preference table* (PCPT) that specifies preferences on X_i, conditioned on values of ancestor issues in the path. These tables make up for the lost structure of T as different ways in which ancestor issues evaluate correspond to different locations in the original tree T. Moreover, missing entries in PCPT of X_i imply equivalence (or indifference) between values of X_i under conditions that do not appear in the PCPT. Clearly, the PLP-tree in Figure 6.2a is collapsible, and can be represented compactly as a single-path tree with nodes labeled by issues in the permutation and PCPTs (cf. Figure 6.2b). Such a collapsed path labeled by issues is sometimes denoted as a sequence of issues in $\hat{R}$ connected by $\triangleright$, e.g., $X_2 \triangleright X_3 \triangleright X_1$ for the path in Figure 6.2b.

Collapsible PLP-trees represented by a single path of nodes will be referred to as *unconditional importance* trees or *UI* trees, for short. The name reflects the fact that the order in which we consider issues when seeking the rank of an outcome is always the same (not conditioned on the values of ancestor issues of higher importance).

Let L be a collapsible PLP-tree. If for every path in L the order of issues labeling the path is exactly $\hat{R}$, and L has the same preference $1 > 0$ on *every* node, then every PCPT in the collapsed tree contains the same preference $1 > 0$, no matter the evaluation of the ancestor issues. Thus, every PCPT in the collapsed form can be simplified to a single *fixed* preference $1 > 0$, a shorthand for its full-sized counterpart. We call the resulting collapsed tree a *UI* tree with *fixed preferences*, or a *UI-FP* PLP-tree.

A similar simplification is possible if every path in L has the same ordering of issues which again is exactly $\hat{R}$, and for every issue X_i all nodes in L labeled with X_i have the same preference on values of X_i (either $1_i > 0_i$ or $0_i > 1_i$). Such collapsed trees are called *UI-UP* PLP-trees, with *UP* standing for *unconditional preference*.

As an example, the *UI-UP* tree in Figure 6.3b is the collapsed representation of the collapsible tree in Figure 6.3a.

In all other cases, we refer to collapsed PLP-trees as *UI-CP* PLP-trees, with *CP* standing for *conditional preference*. If preferences on an issue in such a tree depend in an essential way on all preceding issues, there is no real saving in the size of representation (instead of an exponential PLP-tree we have a small tree but with preference tables that are of exponential size). However, if the preference on an issue depends only on a few higher importance issues say, never more than one or two (or, more generally, never more than some fixed bound b), the collapsed representation is significantly smaller.

As an aside, we note that not every path of nodes labeled with issues and PCPTs is a *UI* tree. An example is given in Figure 5.1e. Indeed, one can see that there is no PLP-tree that would collapse to it. There is a simple condition characterizing paths with nodes labeled with issues and PCPTs that are valid *UI* trees. This matter is not essential to our discussion later on and we will not discuss it further here due to the space limit.

When a PLP-tree is not collapsible, the importance of an issue depends on where it is located in the tree. We will refer to such PLP-trees as *conditional importance* trees or *CI* trees.

Let T be a *CI* PLP-tree. We call T a *CI-FP* tree if every non-leaf node in T is labeled by an issue with preference $1 > 0$. An example of a *CI-FP* PLP-tree is shown in Figure 5.1f, where preferences on each non-leaf node are $1 > 0$ and hence omitted. If, for every issue X_i, all nodes in T labeled with X_i have the same preference ($1_i > 0_i$ or $0_i > 1_i$) on X_i, we say T is a *CI-UP* PLP-tree. All other non-collapsible PLP-trees are called *CI-CP* PLP-trees.

5.3 Passive Learning

An *example* is a tuple (α, β, v), where α and β are two *distinct* outcomes from combinatorial domain $\mathcal{X}$ over a set $\mathcal{I} = \{X_1, \ldots, X_p\}$ of binary issues, and $v \in \{0, 1\}$. An example $(\alpha, \beta, 1)$ states that α is strictly preferred to β ($\alpha \succ \beta$). Similarly, an example $(\alpha, \beta, 0)$ states that α and β are equivalent ($\alpha \approx \beta$). Let $\mathcal{E} = \{e_1, \ldots, e_m\}$ be a set of examples over $\mathcal{I}$, with $e_i = (\alpha_i, \beta_i, v_i)$. We set $\mathcal{E}^{\approx} = \{e_i \in \mathcal{E} : v_i = 0\}$, and $\mathcal{E}^{\succ} = \{e_i \in \mathcal{E} : v_i = 1\}$. In the following, we denote by p and m the number of issues and the number of examples, respectively.

For a PLP-tree T in full representation we denote by $|T|$ the size of T, that is, the number of nodes in T. If T stands for a *UI* tree, we write $|T|$ for the size of T measured by the total size of preference tables associated with issues in T. The size of a preference table is the total size of preferences in it, each preference measured as the number of values in the condition plus 1 for the preferred value in the domain of the issue. In particular, the sizes of *UI-FP* and *UI-UP* trees are given by the number of nodes on the path.

A PLP-tree T *satisfies* an example e if T orders the two outcomes of e in the same way as they are ordered in e. Otherwise, T *falsifies* e. Formally, T *satisfies* $e = (\alpha, \beta, 1)$ if $\alpha \succ_T \beta$, and T *satisfies* $e = (\alpha, \beta, 0)$ if $\alpha \approx_T \beta$. We say T is *consistent* with a set $\mathcal{E}$ of examples if T satisfies every example in $\mathcal{E}$.

In this work, we study the following passive learning problems for PLP-trees of all types we introduced.

Definition 32. Consistent-learning (ConsLearn): given an example set $\mathcal{E}$, decide whether there exists a PLP-tree T (of a particular type) such that T is consistent with $\mathcal{E}$.

Definition 33. Small-learning (SmallLearn): given an example set $\mathcal{E}$ and a positive integer l ($l \leq |\mathcal{E}|$), decide whether there exists a PLP-tree T (of a particular

type) such that T is consistent with $\mathcal{E}$ and $|T| \leq l$.

Definition 34. Maximal-learning (MaxLearn): given an example set $\mathcal{E}$ and a positive integer k ($k \leq m$), decide whether there exists a PLP-tree T (of a particular type) such that T satisfies at least k examples in $\mathcal{E}$.

5.4 Learning UI PLP-trees

In this section, we study the passive learning problems for collapsible PLP-trees in their collapsed representations as *UI-FP*, *UI-UP* and *UI-CP* trees.

The ConsLearn Problem

The ConsLearn problem is in the class P for *UI-FP* and *UI-UP* trees. To show it, we present a general template of an algorithm that learns a *UI* tree. Next, for each of the classes *UI-FP* and *UI-UP*, we specialize the template to a polynomial-time algorithm.

The template algorithm is shown as Algorithm 1. The input consists of a set $\mathcal{E}$ of examples and a set $\mathcal{I}$ of issues from which node labels can be selected. Throughout the execution, the algorithm maintains a set S of unused issues, initialized to $\mathcal{I}$, and a set of examples that are not yet ordered by the tree constructed so far.

If the set of strict examples is empty, the algorithm returns an empty tree. Otherwise, the algorithm identifies the set $AI(\mathcal{E}, S)$ of issues in S that are *available* for selection as the label for the next node. If that set is empty, the algorithm terminates with failure. If not, an issue, say X_l, is selected from $AI(\mathcal{E}, S)$, and a PCPT for that issue is constructed. Then the sets of examples not ordered yet and of issues not used yet are updated, and the steps repeat.

To obtain a learning algorithm for a particular class of *UI* trees (*UI-FP* or *UI-UP*) we need to specify the notion of an available issue (needed for line 3) and describe how to construct a partial conditional preference table (needed for line 8).

Algorithm 1: Procedure *learnUI* that learns a UI tree

Input: $\mathcal{E}$ and $S = \mathcal{I}$
Output: A sequence T of issues from $\mathcal{I}$ and PCPTs that define a *UI* tree consistent with $\mathcal{E}$, or FAILURE if such a tree does not exist

```
T ← empty sequence;
while E^≻ ≠ ∅ do
    Construct AI(E, S);
    if AI(E, S) = ∅ then
        return FAILURE;
    end
    X_l ← an element from AI(E, S);
    Construct PCPT(X_l);
    T ← T, (X_l, PCPT(X_l));
    E ← E\{e ∈ E^≻ : e is decided on X_l};
    S ← S\{X_l};
end
return T;
```

To this end, let us define $NEQ(\mathcal{E}, S)$ to be the set of all issues in S (where $S \subseteq \mathcal{I}$) that incorrectly handle at least one equivalent example in $\mathcal{E}^{\approx}$. That is, for an issue $X \in S$ we have $X \in NEQ(\mathcal{E}, S)$ precisely when for some example $(\alpha, \beta, 0)$ in $\mathcal{E}$, $\alpha(X) \neq \beta(X)$. Similarly, let us define $EQ(\mathcal{E}, S)$ to be the set of issues in S that do not order any of the strict examples in $\mathcal{E}$. That is, for an issue $X \in S$ we have $X \in EQ(\mathcal{E}, S)$ precisely when for every example $(\alpha, \beta, 1)$ in $\mathcal{E}$, $\alpha(X) = \beta(X)$.

Fixed Preferences. For the problem of learning *UI-FP* trees, we define $AI(\mathcal{E}, S)$ to contain every issue $X \notin NEQ(\mathcal{E}, S)$ such that

(1) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$.

Proposition 2. *If there is a UI-FP tree consistent with all examples in $\mathcal{E}$ and using only issues from S as labels, then an issue $X \in S$ is a top node of some such tree if and only if $X \in AI(\mathcal{E}, S)$.*

Proof. Let T be a *UI* tree consistent with $\mathcal{E}$ and having only issues from S as labels. Let X be the issue labeling the top node of T. Clearly, $X \notin NEQ(\mathcal{E}, S)$, as otherwise, T would strictly order two outcomes α and β such that $(\alpha, \beta, 0) \in \mathcal{E}^{\approx}$. To prove

condition (1), let us consider any example $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$. Since T is consistent with $(\alpha, \beta, 1)$, $\alpha(X) \geq \beta(X)$. Consequently, $X \in AI(\mathcal{E}, S)$.

Conversely, let $X \in AI(\mathcal{E}, S)$ and let T be a *UI-FP* tree consistent with all examples in $\mathcal{E}$ and using only issues from S as labels (such a tree exists by assumption). If X labels the top node in T, we are done. Otherwise, let T' be a tree obtained from T by adding at the top of T another node, labeling it with X and removing from T the node labeled by X, if such a node exists. By the definition of $AI(\mathcal{E}, S)$ we have that $X \notin NEQ(\mathcal{E}, S)$ and that condition (1) holds for X. Using these properties, one can show that T' is also a *UI-FP* tree consistent with all examples in $\mathcal{E}$. Since the top node of T' is labeled by X, the assertion follows. □

We now specialize Algorithm 1 by using in line 3 the definition of $AI(\mathcal{E}, S)$ given above and by setting each $PCPT(X_l)$ to the fixed unconditional preference $1_l > 0_l$. Proposition 2 directly implies the correctness of this version of Algorithm 1.

Theorem 8. *Let $\mathcal{E}$ be a set of examples over a set $\mathcal{I}$ of binary issues. Algorithm 1 adjusted as described above terminates and outputs a sequence T representing a UI-FP tree consistent with $\mathcal{E}$ if and only if such a tree exists.*

We note that issues in $NEQ(\mathcal{E}, S)$ are never used when constructing $AI(\mathcal{E}, S)$. Thus, in the case of *UI-FP* trees, S could be initialized to $\mathcal{I} \setminus NEQ(\mathcal{E}, \mathcal{I})$. In addition, if an issue selected for the label of the top node belongs to $EQ(\mathcal{E}^{\succ}, S)$, it does not in fact decide any of the strict examples in $\mathcal{E}$ and can be dropped. The resulting tree is also consistent with all the examples. Thus, the definition of $AI(\mathcal{E}, S)$ can be refined by requiring one more condition: $X \notin EQ(\mathcal{E}^{\succ}, S)$. That change does not affect the correctness of the algorithm but eliminates a possibility of generating trees with "redundant" levels.

Unconditional Preferences. The case of learning *UI-UP* trees is very similar to the previous one. Specifically, we define $AI(\mathcal{E}, S)$ to contain an issue $X \in S$ precisely

when $X \notin NEQ(\mathcal{E}, S)$ and

(2) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$, or for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \leq \beta(X)$.

We obtain an algorithm learning *UI-UP* trees by using in line 3 the present definition of $AI(\mathcal{E}, S)$. In line 8, we take for $PCPT(X_l)$ either $1_l > 0_l$ or $0_l > 1_l$ (depending on which of the two cases in (2) holds for X_l).

The correctness of this algorithm follows from a property similar to that in Proposition 2.

As in the previous case, here too S could be initialized to $\mathcal{I} \setminus NEQ$, and the condition $X \notin EQ(\mathcal{E}^{\succ}, S)$ could be added to the definition of $AI(\mathcal{E}, S)$.

Conditional Preferences. The problem is in NP because one can show that if a *UI-CP* tree consistent with $\mathcal{E}$ exists (*a priori*, it does not have to have size polynomial in the size of $\mathcal{E}$), then another such tree of size polynomial in the size of $\mathcal{E}$ exists, as well. We conjecture that the general problem of learning *UI-CP* trees is, in fact, NP-complete. As we have only partial results for this case, the study of the *UI-CP* tree learning will be the subject of future work.

The SmallLearn Problem

Algorithm 1 produces a *UI* PLP-tree consistent with $\mathcal{E}$, if one exists. In many cases, it is desirable to compute a small, sometimes even the smallest, representation consistent with $\mathcal{E}$. We show that these problems for *UI* trees are NP-hard.

Theorem 9. *The* SmallLearn *problem is NP-complete for each class of* $\{UI\} \times \{FP, UP, CP\}$.

Proof. We present the proof only in the case of *UI-FP*. The argument in other cases (*UI-UP* and *UI-CP*) is similar.

(Membership) One can guess a *UI-FP* PLP-tree T in linear time, and verify in polynomial time that T has at most l issues and satisfies every example in $\mathcal{E}$.

(Hardness) We present a polynomial-time reduction from the *hitting set problem* (HSP), which is NP-complete [39]. To recall, in HSP we are given a finite set $U = \{a_1, \ldots, a_n\}$, a collection $C = \{S_1, \ldots, S_d\}$ of subsets of U with $\bigcup_{S_i \in C} S_i = U$, and a positive integer $k \leq n$, and the problem is to decide whether U has a hitting set U' such that $|U'| \leq k$ ($U' \subseteq U$ is a *hitting* set for C if $U' \cap S_i \neq \emptyset$ for all $S_i \in C$). Given an instance of HSP, we construct an instance of our problem as follows.

1. $\mathcal{I} = \{X_i : a_i \in U\}$ (thus, $p = n$).

2. $\mathcal{E} = \{(\mathbf{s}_i, \mathbf{0}, 1) : S_i \in C\}$, where $\mathbf{s}_i$ is a p-bit vector such that $\mathbf{s}_i[j] = 1 \Leftrightarrow a_j \in S_i$ and $\mathbf{s}_i[j] = 0 \Leftrightarrow a_j \notin S_i$ $(1 \leq j \leq p)$, and $\mathbf{0}$ is a p-bit vector of all 0's (thus, $m = d$).

3. We set $l = k$.

We need to show that U has a hitting set of size at most k if and only if there exists a *UI-FP* PLP-tree of size at most l consistent with $\mathcal{E}$.

($\Rightarrow$) Assume U has a hitting set U' of size k. Let U' be $\{a_{j_1}, \ldots, a_{j_k}\}$. Define a *UI-FP* PLP-tree $L = X_{j_1} \triangleright \ldots \triangleright X_{j_k}$. We show that L is consistent with $\mathcal{E}$. Let $e = (\alpha_e, \beta_e, 1)$ be an arbitrary example in $\mathcal{E}$, where $\alpha_e = \mathbf{s}_i$ and $\beta_e = \mathbf{0}$. Since U' is a hitting set, there exists r, $1 \leq r \leq k$, such that $a_{j_r} \in S_i$. Thus, there exists r, $1 \leq r \leq k$, such that $\alpha_e(X_{j_r}) = 1$. Let r be the smallest with this property. It is clear that e is decided at X_{j_r}; thus, we have $\alpha_e \succ_L \beta_e$.

($\Leftarrow$) Assume there is a *UI-FP* PLP-tree L of l issues in $\mathcal{I}$ such that L is consistent with $\mathcal{E}$. Moreover, we assume $L = X_{j_1} \triangleright \ldots \triangleright X_{j_l}$. Let $U' = \{a_{j_1}, \ldots, a_{j_l}\}$. We show by means of contradiction. Assume that U' is not a hitting set. That is, there exists a set $S_i \in C$ such that $U' \cap S_i = \emptyset$. Then, there exists an example $e = (\alpha_e, \beta_e, 1)$, where $\alpha_e = \mathbf{s}_i$ and $\beta_e = \mathbf{0}$, such that $\alpha_e \approx_L \beta_e$ because none of the issues $\{X_i : \alpha_e(X_i) = 1\}$ show up in L. This is a contradiction! Thus, U' is a hitting set. □

Corollary 10. *Given a set $\mathcal{E}$ of examples $\{e_1, \ldots, e_m\}$ over $\mathcal{I} = \{X_1, \ldots, X_p\}$, finding the smallest PLP-tree in each class of $\{UI\} \times \{FP, UP, CP\}$ consistent with $\mathcal{E}$ is NP-hard.*

Consequently, it is important to study fast heuristics that aim at approximating trees of optimal size. Here, we propose a greedy heuristic for Algorithm 1. In every iteration the heuristic selects the issue $X_l \in AI(\mathcal{E}, S)$ that decides the most examples in $\mathcal{E}^{\succ}$. However, for some dataset the resulting greedy algorithm does not perform well: the ratio of the size of the tree computed by our algorithm to the size of the optimal sequence may be as large as $\Omega(p)$. To see this, we consider the following input.

$(1_1 0_2 0_3 0_4, 0_1 0_2 0_3 0_4, 1)$
$(1_1 1_2 0_3 0_4, 0_1 0_2 0_3 0_4, 1)$
$(1_1 0_2 1_3 0_4, 0_1 0_2 0_3 0_4, 1)$
$(0_1 0_2 0_3 1_4, 1_1 0_2 0_3 0_4, 1)$

For each class of $\{UI\} \times \{FP, UP\}$, Algorithm 1 in the worst case computes $X_2 \triangleright X_3 \triangleright X_4 \triangleright X_1$, whereas the optimal tree is $X_4 \triangleright X_1$ (with the PCPTs omitted as they contain only one preference and so, they do not change the asymptotic size of the tree). This example generalizes to the arbitrary number p of issues. Thus, the greedy algorithm to learn small *UI* trees is no better than any other algorithm in the worst case.

Approximating HSP has been extensively studied over the last decades. It has been shown [61] that, unless $NP \subset DTIME(n^{poly \log n})$, HSP cannot be approximated in polynomial time within factor of $c \log n$, where $0 < c < \frac{1}{4}$ and n is the number of elements in the input. The reduction we used above shows that this result carries over to our problem.

Theorem 11. *Unless $NP \subset DTIME(n^{poly \log n})$, the problem of finding the smallest PLP-tree in each class of $\{UI\} \times \{FP, UP, CP\}$ consistent with $\mathcal{E}$ cannot be approximated in polynomial time within factor of $c \log p$, where $0 < c < \frac{1}{4}$.*

It is an open problem whether this result can be strengthened to a factor linear in p (cf. the example for the worst-case behavior of our simple greedy heuristic).

The MaxLearn Problem

When there is no *UI* PLP-tree consistent with the set of all examples, it may be useful to learn a *UI* PLP-tree satisfying as many examples as possible. We show this problem is in fact NP-hard for all three classes of *UI* trees.

Theorem 12. *The* MaxLearn *problem is NP-complete for each class of* $\{UI\} \times \{FP, UP, CP\}$.

Sketch. The problem is in NP. This is evident for the case of *UI-FP* and *UI-UP* trees. If $\mathcal{E}$ is a given set of examples, and k a required lower bound on the number of examples that are to be correctly ordered, then witness trees in these classes (trees that correctly order at least k examples in $\mathcal{E}$) have size polynomial in the size of $\mathcal{E}$. Thus, verification can be performed in polynomial time. For the case of *UI-CP* trees, one can show that if there is a *UI-CP* tree correctly ordering at least k examples in $\mathcal{E}$, then there exists such tree of size polynomial in $|\mathcal{E}|$.

The hardness part follows from the proof in the setting of learning lexicographic strategies [70], adapted to the case of *UI* PLP-trees. □

Corollary 13. *Given a set* $\mathcal{E}$ *of examples* $\{e_1, \ldots, e_m\}$ *over* $\mathcal{I} = \{X_1, \ldots, X_p\}$*, finding a PLP-tree in each class of* $\{UI\} \times \{FP, UP, CP\}$ *satisfying the maximum number of examples in* $\mathcal{E}$ *is NP-hard.*

5.5 Learning CI PLP-trees

Finally, we present results on the passive learning problems for PLP-trees in classes $\{CI\} \times \{FP, UP, CP\}$. We recall that these trees assume full (non-collapsed) representation.

The ConsLearn Problem

We first show that the ConsLearn problem for class *CI-UP* is NP-complete. We then propose polynomial-time algorithms to solve the ConsLearn problem for the classes *CI-FP* and *CI-CP*.

Theorem 14. *The* ConsLearn *problem is NP-complete for class CI-UP.*

Sketch. The problem is in NP because the size of a witness, a *CI-UP* PLP-tree consistent with $\mathcal{E}$, is bounded by $|\mathcal{E}|$ (one can show that if a *CI-UP* tree consistent with $\mathcal{E}$ exists, then it can be modified to a tree of size no larger than $O(|\mathcal{E}|)$). Hardness follows from the proof by Booth et al. [16] showing ConsLearn is NP-hard in the setting of LP-trees. □

For the two other classes of trees, the problem is in P. This is demonstrated by polynomial-time Algorithm 2 adjusted for both classes.

Algorithm 2: The recursive procedure *learnCI* that learns a CI PLP-tree

Input: $\mathcal{E}$, $S = \mathcal{I}$, and t: an unlabeled node
Output: A CI PLP-tree over S consistent with $\mathcal{E}$, or FAILURE

1 **if** $\mathcal{E}^{\succ} = \emptyset$ **then**
2 | Label t as a leaf and **return**;
3 **end**
4 Construct $AI(\mathcal{E}, S)$;
5 **if** $AI(\mathcal{E}, S) = \emptyset$ **then**
6 | **return** FAILURE and terminate;
7 **end**
8 Label t with tuple (X_l, x_l) where X_l is from $AI(\mathcal{E}, S)$, and x_l is the preferred value on X_l;
9 $\mathcal{E} \leftarrow \mathcal{E} \backslash \{e \in \mathcal{E}^{\succ} : e \text{ is decided on } X_l\}$;
10 $S \leftarrow S \backslash \{X_l\}$;
11 Create two edges u_l, u_r and two unlabeled nodes t_l, t_r such that $u_l = \langle t, t_l \rangle$ and $u_r = \langle t, t_r \rangle$;
12 $\mathcal{E}_l \leftarrow \{e \in \mathcal{E} : \alpha_e(X_j) = \beta_e(X_j) = x_l\}$;
13 $\mathcal{E}_r \leftarrow \{e \in \mathcal{E} : \alpha_e(X_j) = \beta_e(X_j) = \overline{x_l}\}$;
14 *learnCI*$(\mathcal{E}_l, S, t_l)$;
15 *learnCI*$(\mathcal{E}_r, S, t_r)$;

Fixed Preference. For class *CI-FP*, we define $AI(\mathcal{E}, S)$ to contain issue $X \notin NEQ(\mathcal{E}, S)$ if

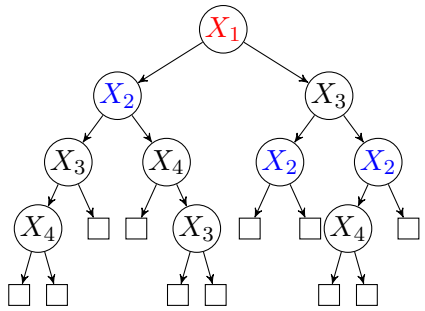
(3) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$.

Proposition 3. *If there is a CI-FP tree consistent with all examples in $\mathcal{E}$ and using only issues from S as labels, then an issue $X \in S$ is a top node of some such tree if and only if $X \in AI(\mathcal{E}, S)$.*

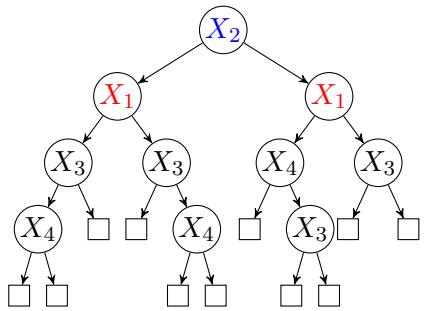
Proof. It is clear that if there exists a *CI-FP* PLP-tree consistent with $\mathcal{E}$ and only using issues from S as labels, then the fact that $X \in S$ labels the root of some such tree implies $X \in AI(\mathcal{E}, S)$.

Now we show the other direction. Let T be the *CI-FP* tree over a subset of S consistent with $\mathcal{E}$, X be an issue such that $X \in AI(\mathcal{E}, S)$. If X is the root issue in T, we are done. Otherwise, we construct a *CI-FP* tree T' by creating a root, labeling it with X, and make one copy of T the left subtree of T' (T'_l) and another, the right subtree of T' (T'_r). For a node t and a subtree B in T, we write t'_l and B'_l, respectively, for the corresponding node and subtree in T'_l. We define t'_r and B'_r similarly. If X does not appear in T, we are done constructing T'; otherwise, we update T' as follows.
1). For every node $t \in T$ labeled by X such that t has two leaf children, we replace the subtrees rooted at t'_l and t'_r in T'_l and T'_r with leaves.
2). For every node $t \in T$ labeled by X such that t has one leaf child and a non-leaf subtree B, we replace the subtree rooted at t'_l in T'_l with B'_l, and the subtree rooted at t'_r in T'_r with a leaf, if $t \in T$ has a right leaf child; otherwise, we replace the subtree rooted at t'_l in T'_l with a leaf, and the subtree rooted at t'_r in T'_r with B'_r.
3). Every other node $t \in T$ labeled by X has two non-leaf subtrees: left non-leaf subtree BL and right BR. For every such node $t \in T$, we replace the subtree rooted at t'_l in T'_l with BL'_l, and the subtree rooted at t'_r in T'_r with BR'_r.

As an example, this construction of T' from T is demonstrated in Figure 5.2. One can show that this construction results in a *CI-CP* tree consistent with $\mathcal{E}$ and, clearly,



(a) T



(b) T'

Figure 5.2: $X_2 \in AI(\mathcal{E}, S)$ is picked at the root

it has its root labeled with X. Thus, the assertion follows. □

Proposition 3 clearly implies the correctness of Algorithm 2 with $AI(\mathcal{E}, S)$ defined as above for class *CI-FP* and each $x_l \in (X_l, x_l)$ set to 1.

Theorem 15. *Let $\mathcal{E}$ be a set of examples over a set $\mathcal{I}$ of binary issues. Algorithm 2 adjusted as described above terminates and outputs a CI-FP tree T consistent with $\mathcal{E}$ if and only if such a tree exists.*

Conditional Preference. For class *CI-CP*, we define that $AI(\mathcal{E}, S)$ contains issue $X \notin NEQ(\mathcal{E})$ if

(4) for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \geq \beta(X)$, or for every $(\alpha, \beta, 1) \in \mathcal{E}^{\succ}$, $\alpha(X) \leq \beta(X)$.

We obtain an algorithm learning *CI-CP* trees by using in line 4 the present definition of $AI(\mathcal{E}, S)$. In line 8, we take for x_l either 1 or 0 (depending on which of the two cases in (4) holds for X_l). The correctness of this algorithm follows from a property similar to that in Proposition 3.

The SmallLearn and MaxLearn Problems

Due to space limits, we only outline the results we have for this case. Both problems for the three *CI* classes are NP-complete. They are in NP since if a witness PLP-tree exists, one can modify it so that its size does not exceed the size of the input. Hardness of the SmallLearn problem for *CI* classes follows from the proof of Theorem 9, whereas the hardness of the MaxLearn problem for *CI* cases follows from the proof by Schmitt and Martignon [70].

5.6 Conclusions

We proposed a preference language, *partial lexicographic preference trees*, *PLP-trees*, as a way to represent preferences over combinatorial domains. For several natural classes of PLP-trees, we studied passive learning problems: ConsLearn, SmallLearn and MaxLearn. All complexity results we obtained are summarized in tables in Table 5.1. The ConsLearn problem for *UI-CP* trees is as of now unsettled. While we are aware of subclasses of *UI-CP* trees for which polynomial-time algorithms are possible, we conjecture that in general, the problem is NP-complete.

	FP	UP	CP
UI	P	P	*NP*
CI	P	NPC	P

(a) ConsLearn

	FP	UP	CP
UI	NPC	NPC	NPC
CI	NPC	NPC	NPC

(b) SmallLearn & MaxLearn

Table 5.1: Complexity results for passive learning problems

For the future research, we will develop good heuristics for our learning algorithms. We will implement these algorithms handling issues of, in general, finite domains of values, and evaluate them on both synthetic and real-world preferential datasets. With PLP-trees of various classes learned, we will compare our models with the ones learned through other learning approaches on predicting new preferences.

Chapter 6 Empirical Study of Learning PLP-Trees and Forests of PLP-Trees

Partial lexicographic preference trees, or *PLP-trees*, is an intuitive formalism that allows compact representation of qualitative preferences over combinatorial domains. In this work, we review different classifications of PLP-trees, and introduce *partial lexicographic preference forests*, or *PLP-forests*, to reduce the high variance of PLP-trees. To support experimentation, we have constructed a preference learning library from existing reservoir of classification datasets. To this end, we implemented an *exact* learner using *answer-set programming*, and an *approximation* learner using a straightforward greedy heuristic. Our empirical results on PLP-trees show that the language offers high accuracy, mostly higher than decision trees when training samples are small. For PLP-forests, we have observed improvements, significant in some cases, from single tree models.

6.1 Introduction

Preferences are everywhere and have been extensively studied by researchers and scientists in artificial intelligence, psychology, operations research, and social choice theory.

Learning to predict preferences has been an important task across many areas in computer science. In label ranking, the task is to learn a model that maps a given outcome to a preference ranking of the labels. In classification, the problem is to apprximate a function by learning a hypothesis that maps an instance to a classifier; in the preference learning setting, the instance would be two outcomes, and the classifier could tell the preference between the two. Besides this machine learning perspective, researchers have studied preference learning with assumptions of the

representations of the model to be learned. Representations of intuitive meaning and compact size are especially of interest. Conditional preference networks (CP-nets)[**?**] is such a formalism, and learning CP-nets have received tremendous attention.

Continuing this line of research on representation-based preference learning, AI researchers have also proposed and studied preference languages exploiting the idea of ordering outcomes lexicographically, such as lexicographic strategies[70], conditional lexicographic trees[21], lexicographic preference trees[16], and partial lexicographic preference trees (PLP-trees)[59].

In this work, we focus on the representations of PLP-trees and forests of PLP-trees. We consider four classes of PLP-trees and their corresponding forests: *unconditional importance and unconditional preference* (UIUP), *unconditional importance and conditional preference* (UICP), *conditional importance and unconditional preference* (CIUP), and *conditional importance and conditional preference* (CICP). We are given a dataset of pairwise preferences, called *examples*. First, for each class, we want to learn a PLP-tree that is in agreement with as many examples in the *training* set as possible, and test it using the *testing* set to see how well the model generalizes to the testing data. Second, for each class, we want to learn a set of PLP-trees, or a forest, hoping for better performance than a single tree. To support our experiments, inspired by the work by Bräuning and Eyke[21], we have built a preference learning library of semi-real-world datasets from classification repositories.

The model of decision trees in machine learning is different from PLP-trees. A decision tree classifies whether an outcome is better than another, but its size usually grows as the training examples, and it does not directly answer optimal queries. Some classes of PLP-trees (e.g., UIUP) have size linear in the size of the attribute domains. Computing an optimal outcome given a PLP-tree takes polynomial time in the size of these domains. Moreover, PLP-trees are arguably more intuitive and more cognitively plausible than decision trees.

We outline the rest of the paper as follows. We will review PLP-trees and its classifications in Section 2, and introduce and define PLP-forests in Section 3. Next, in Section 4, we will discuss the datasets in the preference learning library built by us, including where the original datasets were from, and how we constructed preferencial datasets for our own use. In Section 5, we will present and analyze experimental results obtained by us for learning both PLP-trees and PLP-forests. Finally, we will conclude with a brief summary of our work and a look into possible directions of future work.

6.2 Partial Lexicographic Preference Trees

PLP-trees was originally defined with respect to binary attributes[59]. In this work, we expand it to the general case where attributes are multi-valued. Let $\mathcal{A} = \{X_1, \ldots, X_p\}$ be a set of attributes, with each X_i having its finite domain D_i, the size of which is bounded by a constant. The corresponding *combinatorial domain* over $\mathcal{A}$ is the Cartesian product $CD(\mathcal{A}) = D_1 \times \ldots \times D_p$. Elements in $CD(\mathcal{A})$ are called *outcomes*.

A PLP-tree over $CD(\mathcal{A})$ is a labeled tree, where every non-leaf node is labeled by an attribute from $\mathcal{A}$ and by a preference entry (a total order over D_i), where every leaf node is denoted by a box $\square$, and where non-leaf node has as many outgoing edges, ordered from left to right according to the preference entry, as there are values in the domain of the attribute labeling it. Moreover, we require that on every path from the root to a leaf each attribute appears *at most* once.

To specify the total preorder on outcomes defined by a PLP-tree T, let us enumerate leaves of T from left to right, assigning them integers $1, 2$, etc. For every outcome α, we find its leaf in T by starting at the root of T and proceeding downward. When at a node labeled with an attribute X, we descend to some child of that node based on the value $\alpha(X)$ of the attribute X in α and on the preference entry assigned to that node. If $\alpha(X)$ is the i-th preferred value, we descend to the i-th child. The integer

assigned to the leaf that we eventually get to is the *rank* of α in T, written $r_T(\alpha)$. The preorder $\succeq_T$ on distinct outcomes determined by T is defined as follows: $\alpha \succeq_T \beta$ if $r_T(\alpha) \leq r_T(\beta)$ (smaller ranks are "better"). We also define derived relations $\succ_T$ (strict order) and $\approx_T$ (equivalence or indifference): $\alpha \succ_T \beta$ if $\alpha \succeq_T \beta$ and $\beta \not\succeq_T \alpha$, and $\alpha_T \approx_T \beta$ if $\alpha \succeq_T \beta$ and $\beta \succeq_T \alpha$. Clearly, $\succeq_T$ is a total preorder on outcomes partitioning them into strictly ordered clusters of equivalent outcomes.

To illustrate the notions just introduced, we consider preference orderings of cars over six multi-valued attributes. The *BodyType* (X_1) can be *minivan* (x_{11}), *sedan* (x_{12}), or *sport* (x_{13}). The *Capacity* (X_2) can be *2* (x_{21}), *5* (x_{22}), or *7-or-more* (x_{23}). The *Color* (X_3) can be *black* (x_{31}) or *white* (X_{32}). The *Make* (X_4) is either *Honda* (x_{41}) or *Ford* (x_{42}). The *Price* (X_5) can be *high* (x_{51}), *low* (X_{52}), or *medium* (X_{53}). Finally, *Transmission* (X_6) can be *automatic* (x_{61}) or *manual* (x_{62}). An agent could specify her preferences over dinners as a PLP-tree T in Figure 6.1. The *BodyType* is the most important attribute to the agent and she prefers minivan, followed by sedan and sport. Her next most important attribute is contingent upon what type of cars the agent is considering. For minivans, her most important attribute is *Make*, for which she likes Honda more than Ford. Among sedans, her most important attribute is *Price*, on which she prefers medium over low, and low over high. Amongst sport cars, her top priority is *transmission* and she prefers manual to automatic.

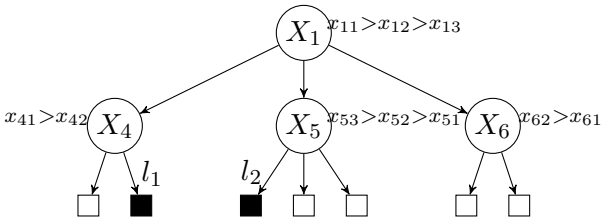
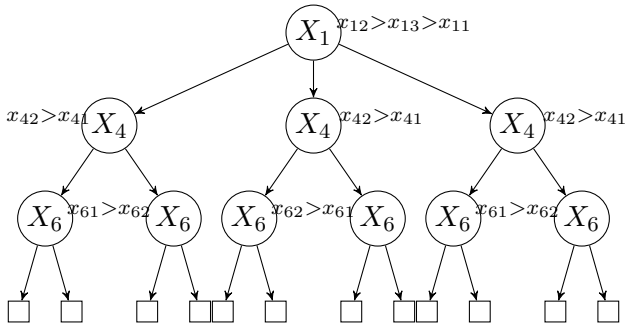


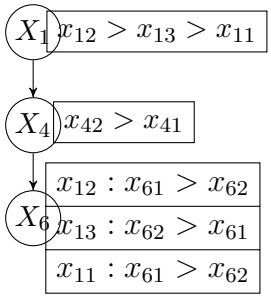
Figure 6.1: PLP-tree T over the car domain

Let us consider two cars. Car c_1 is a white Ford minivan with a capacity of 7 or more, a middle-range price, and an automatic transmission; that is, $c_1 = \langle x_{11}, x_{23}, x_{32}, x_{42}, x_{53}, x_{61} \rangle$. Car c_2 is a black Honda sedan with a capacity of 2,

a middle-range price, and a manual transmission: $c_2 = \langle x_{12}, x_{21}, x_{31}, x_{41}, x_{53}, x_{62} \rangle$. Traversing the tree T, we see that c_1 reaches leaf l_1 of rank 2 and c_2 leaf l_2 of rank 3. We have $r_T(c_1) < r_T(c_2)$. Thus, we see $c_1 \succ_T c_2$.



(a) Collapsible PLP-tree



(b) UICP PLP-tree

Figure 6.2: PLP-trees over the car domain

Classification of PLP-Trees

In the worst case, the size of a PLP-tree is exponential in the number of attributes in $\mathcal{A}$. However, some PLP-trees have a special structure that allows us to "collapse" them and obtain more compact representations. This yields a natural classification of PLP-trees, which we describe below.

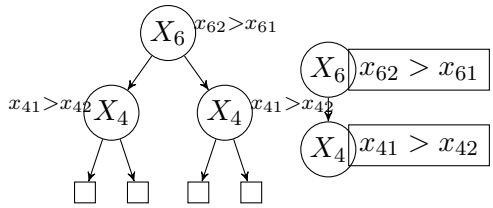
Let $R \subseteq \mathcal{A}$ be the set of attributes that appear in a PLP-tree T. We say that T is *collapsible* if there is a permutation $\hat{R}$ of elements in R such that for every path in T from the root to a leaf, attributes that label nodes on that path appear in the same order in which they appear in $\hat{R}$.

If a PLP-tree T is collapsible, we can represent T by a single path of nodes labeled with attributes according to the order in which they occur in $\hat{R}$, where a node labeled with an attribute X_i is also assigned a *conditional preference table* (CPT) that specifies preferences on X_i, conditioned on values of ancestor attributes in the path. These tables make up for the lost structure of T as different ways in which ancestor attributes evaluate correspond to different locations in the original tree T. Moreover, missing entries in PCPT of X_i imply equivalence (or indifference) between values of X_i under conditions that do not appear in the PCPT. Clearly, the PLP-tree in Figure 6.2a is collapsible, and can be represented compactly as a single-path tree with nodes labeled by attributes in the permutation and PCPTs (cf. Figure 6.2b). Note that the PLP-tree in Figure 6.1 is also collapsible and can be collapsed into a UICP tree.

Collapsible PLP-trees represented by a single path of nodes will be referred to as *unconditional importance* trees or *UI* trees, for short. The name reflects the fact that the order in which we consider attributes when seeking the rank of an outcome is always the same (not conditioned on the values of ancestor attributes of higher importance).

Let L be a collapsible PLP-tree. If every path in L has the same ordering of attributes which again is exactly $\hat{R}$, and for every attribute X_i all nodes in L labeled with X_i have the same preference on values of X_i (either $1_i > 0_i$ or $0_i > 1_i$). Such collapsed trees are called *UI-UP* PLP-trees, with *UP* standing for *unconditional preference*. As an example, the *UI-UP* tree in Figure 6.3b is the collapsed representation of the collapsible tree in Figure 6.3a.

In all other cases, we refer to collapsed PLP-trees as *UI-CP* PLP-trees, with *CP* standing for *conditional preference*. If preferences on an attribute in such a tree depend in an essential way on all preceding attributes, there is no real saving in the size of representation (instead of an exponential PLP-tree we have a small tree but



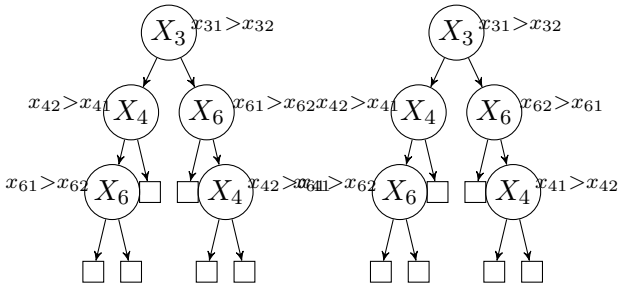
(a) Collapsible PLP-tree (b) UIUP PLP-tree

Figure 6.3: PLP-trees over the car domain

with preference tables that are of exponential size). However, if the preference on an attribute depends only on a few higher importance attributes say, never more than one or two (or, more generally, never more than some fixed bound b), the collapsed representation is significantly smaller.

When a PLP-tree is not collapsible, the importance of an attribute depends on where it is located in the tree. We will refer to such PLP-trees as *conditional importance* trees or *CI* trees.

Let T be a *CI* PLP-tree. We call T a *CI-UP* tree if for every attribute X_i, all nodes in T labeled with X_i have the same preference entry on X_i. All other non-collapsible PLP-trees are called *CI-CP* PLP-trees. Examples are shown in Figure 6.4a and Figure 6.4b.



(a) CIUP PLP-tree (b) CICP PLP-tree

Figure 6.4: CI PLP-trees over the car domain

6.3 Partial Lexicographic Preference Forests

We introduce a new notion of *PLP-forests* that is a collection of PLP-trees. Let F be a PLP-forest such that $F = \{T_1, \ldots, T_n\}$. Let us denote by $N_F(o_1, o_2) = |\{T \in F : o_1 \succ_T o_2\}|$ the number of trees in the forests where the outcome o_1 is preferred to the outcome o_2. Now consider the dominance testing problem. Given a preference forest F, and two outcomes o_1 and o_2, we say that $o_1 \succ_F^{Maj} o_2$ iff $N_F(o_1, o_2) > N_F(o_2, o_1)$, and that $o_1 \approx_F^{Maj} o_2$ iff $N_F(o_1, o_2) = N_F(o_2, o_1)$, where *Maj* stands for the majority rule. Indeed, the majority rule is an intuitive and computationally easy preference aggregation rule. In general, the majority rule may lead to the so-called Condorcet paradox, where the $\succ_F^{Maj}$ relation contains a cycle. However, it is not the case for our datasets. Other possible aggregators are positional scoring rules (adjusted for total *preorders*), Copeland's method, among others.

6.4 Problems and Complexities

In this work, we focus on the MaxLearn problem as follows for PLP-trees.

Definition 35. Maximal-learning (MaxLearn): given an example set $\mathcal{E}$ and a positive integer k ($k \leq |\mathcal{E}|$), decide whether there exists a PLP-tree T (of a particular type) such that T satisfies at least k examples in $\mathcal{E}$.

The MaxLearn problem has been shown NP-complete for all four classes of PLP-trees[59]. The following problems are also of interest.

Definition 36. Best-linearization (BestLin): Given a directed graph $G = (V, E)$, find the best linearization, that is, the total order $\succ$ over V that agrees with the as many edges $uv \in E$ as possible.

This problem is related to solving the MaxLearn problem, when we need to pick a preference entry for an attribute, although it is trivial as we bound the sizes of

the attribute domains by a constant. We now show the complexity of the BestLin problem, if it has not been proved yet.

Theorem 16. *The* BestLin *problem is NP-hard.*

Proof. To show NP-hardness, it suffices to show that the BestLin is at least as hard as another NP-hard problem: the Minimum Feedback Arc Set (MFAS) problem[1]. To solve the BestLin problem, we can first solve the MFAS problem and then apply the topological sorting, taking linear time, on the solution to obtain the best linearization. □

Schmitt and Martegnon[70] proved that, for UIFP trees[2], the greedy heuristic approximates the MaxLearn problem to within a factor of p. We are interested in extending this results to more general classes of trees. Yet another problem we might want to study is the MaxLearn problem in the setting of PLP-forests. That is, given an example set $\mathcal{E}$ and positive integers f and k, decide whether there exists a PLP-forest F of size f that satisfies at least k examples in $\mathcal{E}$.

6.5 Preference Learning Library

Here we describe how we built the datasets[3]to support preference learning experiments. We limited the number of issues to 10, and the size of their domains to 4 if it is more than 4. The description of the datasets in this library are shown in Table 6.1, where we denote by p the number of attributes, $|\mathcal{X}|$ the number of outcomes, $|\mathcal{E}^{\succ}|$ the number of strict examples, and $|\mathcal{E}^{\approx}|$ the number of equivalent examples.

BreastCancerWisconsin The BreastCancerWisconsin dataset has 270 outcomes over 9 attributes. To generate equivalent and strict examples for the dataset, we

[1]Given a directed graph $G = (V, E)$, the Minimum Feedback Arc Set (MFAS) problem is to find the subset $F \subseteq E$ such that $G' = (V, E \backslash F)$ is acyclic and $|F|$ is minimum.

[2]UIFP is subclass of UIUP trees where preference entries for all attributes are fixed.

[3]We will make the library public once the paper is accepted.

assume that outcomes labeled by "benign" are better than those by "malignant." For equivalent examples, we have that outcomes labeled by "benign" are equivalent to one another, so are those labeled by "malignant."

CarEvaluation The CarEvaluation dataset has 1728 outcomes over 6 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "vgood" are better than those by "good," which are better than those by "acc," which are preferred to those by "unacc."

CreditApproval The CreditApproval dataset has 520 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "+" (positive) are better than those by "-" (negative).

GermanCredit The GermanCredit dataset has 914 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "1" (good) are better than those by "2" (bad).

Ionosphere The Ionosphere dataset has 118 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "g" (good) are better than those by "b" (bad).

MammographicMass The MammographicMass dataset has 118 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "0" (benigh) are better than those by "1" (malignant).

Mushroom The Mushroom dataset has 184 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "e" (edible) are better than those by "p" (poisonous).

Nursery The Nursery dataset has 184 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "spec_prior" are better than those by "priority," which are better than those by "very_recom," which are preferred to those by "recommend," which again are better than those by "not_recom."

Dataset	p	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{E}^{\succ}\rvert$	$\lvert\mathcal{E}^{\approx}\rvert$
BreastCancerWisconsin	9	270	9,009	27,306
CarEvaluation	6	1728	682,721	809,407
CreditApproval	10	520	66,079	68,861
GermanCredit	10	914	172,368	244,873
Ionosphere	10	118	3,472	3,431
MammographicMass	5	62	792	1,099
Mushroom	10	184	8,448	8,388
Nursery	8	1,266	548,064	252,681
SPECTHeart	10	115	3,196	3,359
TicTacToe	9	958	207,832	250,571
Vehicle	10	455	76,713	26,572
Wine	10	177	10,322	5,254

Table 6.1: Description of datasets in the library

SPECTHeart The SPECTHeart dataset has 115 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "0" (positive) are better than those by "1" (negative).

TicTacToe The TicTacToe dataset has 958 outcomes over 9 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "positive" are better than those by "negative".

Vehicle The Vehicle dataset has 455 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "bus" are better than those by "opel," which are better than those by "saab," which are preferred to those by "van."

Wine The Wine dataset has 455 outcomes over 10 attributes. To generate equivalent and strict examples for the dataset, we assume that outcomes labeled by "1" are better than those by "2," which are better than those by "3."

6.6 Experimentation

We design and implement exact and approximating systems to evaluate PLP-trees and forests of PLP-trees.

Evaluating PLP-Trees

To evaluate the feasibility of PLP-trees of various classes, we implemented an exact learner using Answer-Set Programming, as well as a greedy learning algorithm.

First, for all datasets we achieve results for UIUP PLP-trees using exact and greedy learning methods, and for decision trees. For dataset D, we randomly pick $R_D \subset \mathcal{E}^{\succ}$ with $1 \leq |R_D| \leq 250$ as the set of *training* examples, and $T_D = \mathcal{E}^{\succ} \setminus R_D$ as the set of *testing* examples. Then, from R_D, we train a UIUP PLP-tree T_X using Answer-Set Programming such that T_X decides the most examples in R_D, and another UIUP PLP-tree T_{MH} using greedy heuristic. To compare our preference trees with decision trees, we also train a decision tree T_{DT} from R_D. Finally, on T_D we test the models T_X, T_{MH} and T_{DT} to compute the percentages of strict examples in T_D that are correctly decided by them. This process is repeated 20 times for all the datasets, and the average results are reported as *learning curves* presented in Figure 6.5a, Figure 6.5b, Figure 6.5c, Figure 6.5d, Figure 6.5e, Figure 6.5f, Figure 6.5g, Figure 6.5h, Figure 6.5i, Figure 6.5j, Figure 6.5k, and Figure 6.5l.

What we have learned from Figure 6.5:

1. For the exact learning method using ASP (X-UIUP), learned UIUP trees achieve over 70% of accuracy across all datasets using small samples of 250 examples: 70%-80%: 3 datasets, 80%-90%: 5 datasets, and 90%-100%: 4 datasets.

2. For the greedy heuristic method (MH-UIUP), t learned UIUP trees achieve also over 70% of accuracy across all datasets using small samples of 250 examples: 70%-80%: 5 datasets, 80%-90%: 4 datasets, and 90%-100%: 3 datasets.

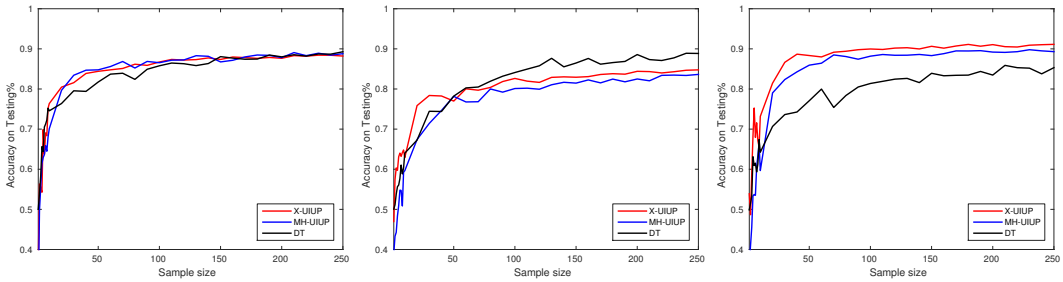
3. For the exact search method used in *fitctree*, a matlab tool for classification decision tree (DT), learned decision trees achieve over 60% of accuracy across all datasets using small samples of 250 examples: 60%-70%: 2 datasets, 70%-80%: 1 datasets, 80%-90%: 6 datasets, and 90%-100%: 3 datasets.

4. Comparing X-UIUP and DT, we observe that X-UIUP outperforms DT on 8 datasets (CreditApproval, GermanCredit, Ionosphere, Nursery, SpectHeart, TicTacToe, Vehicle, and Wine), that DT outperforms X-UIUP on 2 datasets (CarEvaluation and Mushroom), that X-UIUP and DT are close on the other 2 datasets (BreastCancerWisconsin and MammographicMass). Comparing M-UIUP and DT, we observe that MH-UIUP outperforms DT on 7 datasets (CreditApproval, GermanCredit, Nursery, SpectHeart, TicTacToe, Vehicle, and Wine), that DT outperforms MH-UIUP on 2 datasets (CarEvaluation and Mushroom), that MH-UIUP and DT are close on the other 3 datasets (BreastCancerWisconsin, Ionosphere, and MammographicMass). This comparison shows UIUP trees, the simplest PLP-trees, are favored over decision trees in terms of both accuracy and representation intuition, when training samples are of small sizes.

5. Comparing X-UIUP and MH-UIUP, we see that mostly the greedy heuristic is close to the exact algorithm, except for 2 datasets: Ionosphere and Mushroom. This indicates the greedy method works well.

Next, since X-UIUP does not scale well even for small samples of sizes near 250, we have performed experiments with more expansive samples using the greedy heuristic on all four classes of PLP-trees: UIUP, UICP-1, CIUP and CICP. For dataset $\mathcal{E}^{\succ}$, we generate $R_{\mathcal{E}^{\succ}}$ $(1\% * |\mathcal{E}^{\succ}| \leq |R_{\mathcal{E}^{\succ}}| \leq 70\% * |\mathcal{E}^{\succ}|)$ and $T_{\mathcal{E}^{\succ}}$ $(T_{\mathcal{E}^{\succ}} = \mathcal{E}^{\succ} \backslash R_{\mathcal{E}^{\succ}})$ for training and testing, respectively. Then, from $R_{\mathcal{E}^{\succ}}$, we train UIUP, UICP-1, CIUP and CICP trees (T_1, T_2, T_3 and T_4) using the greedy heuristic. Finally, on $T_{\mathcal{E}^{\succ}}$ we test the models T_1, T_2, T_3 and T_4 to compute the percentages of strict examples in $T_{\mathcal{E}^{\succ}}$ that are correctly decided by them. In Table 6.2, we present results of accuracy on testing using 70% of $\mathcal{E}^{\succ}$ in the training phase.

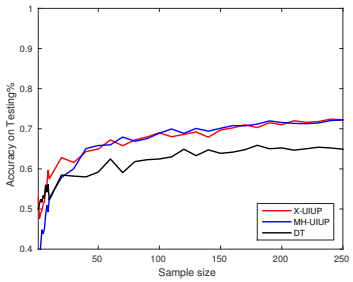
What we have learned from Table 6.2:



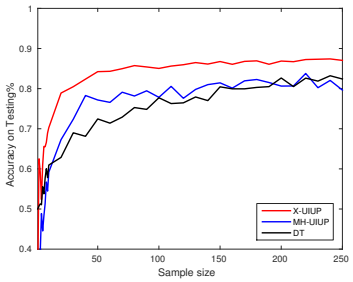
(a) BreastCancerWisconsin

(b) CarEvaluation

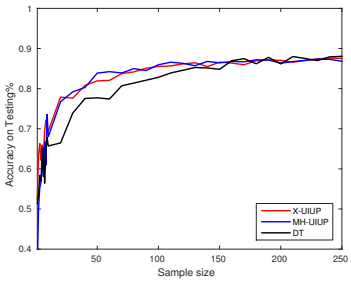
(c) CreditApproval



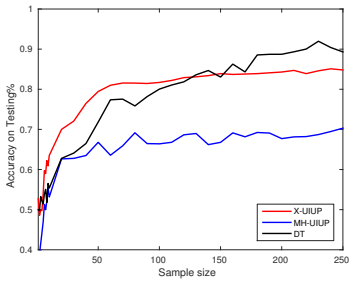
(d) GermanCredit



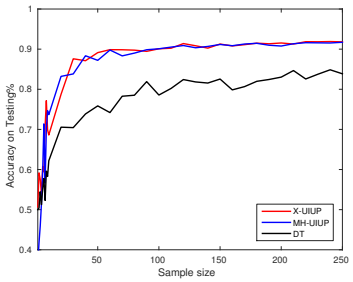
(e) Ionosphere



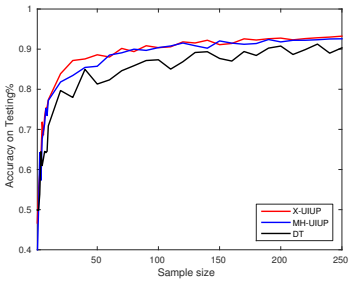
(f) MammographicMass



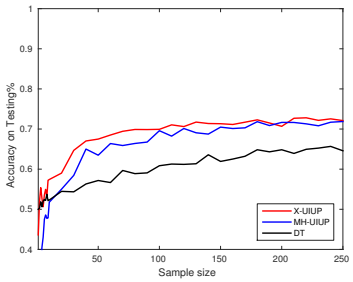
(g) Mushroom



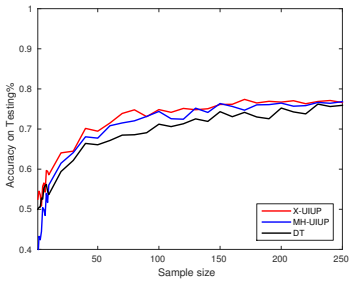
(h) Nursery



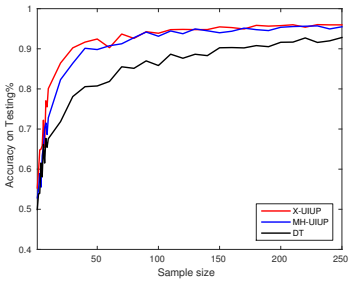
(i) SpectHeart



(j) TicTacToe



(k) Vehicle



(l) Wine

Figure 6.5: UIUP PLP-trees vs. decision trees

1. UIUP trees performs well, with all datasets above 70%, and 4 datasets (GermanCredit, Mushroom, TicTacToe and Vehicle) below 85%.

2. UICP-1 trees outperforms UIUP trees on all but one dataset Ionosphere.

3. CIUP trees outperforms UIUP trees on all but one dataset MammographicMass.

Dataset	UIUP	UICP-1	CIUP	CICP
BreastCancerWisconsin	90.7	91.4	90.7	91.4
CarEvaluation	85.8	86.0	85.9	86.0
CreditApproval	91.4	91.7	92.0	92.2
GermanCredit	74.3	74.6	74.5	75.7
Ionosphere	87.1	[86.9]	88.5	90.4
MammographicMass	88.2	89.5	[86.9]	90.0
Mushroom	71.6	74.2	75.6	76.6
Nursery	92.9	93.0	93.0	93.0
SPECTHeart	93.4	94.9	94.8	95.7
TicTacToe	73.9	74.5	75.4	76.2
Vehicle	79.2	80.4	80.0	81.2
Wine	95.5	97.8	97.5	97.8

Table 6.2: Accuracy percents on the testing data (30% of $\mathcal{E}^{\succ}$) for all four classes of PLP-trees, using models learned by the greedy algorithm from the learning data (the other 70% of $\mathcal{E}^{\succ}$)

4. CICP trees outperforms all other classes of trees on all datasets.

Evaluating PLP-Forests

To further boost up performances, we now show empirical results of learning *PLP-forests*.

First, we show results for UIUP PLP-forests using exact learning and the greedy heuristic. In each experiment, we randomly partition a dataset into training set (70%) and testing set (30%), learn a forest (the size of it indicated by on the x-axis) where each tree is learned from 50 randomly selected examples from the training set, and test the forest against the testing set. We repeat it 20 times and report the averge accuracy (indicated on the y-axis) in the plots. Note that the plots also include a straight line representing the result for learning a UIUP PLP-tree, had we used all the training data to learn a single tree using the greedy heuristic.

What we have learned from Table 6.3:

1. UIUP forests using the greedy method outperforms UIUP trees using the greedy method on all but one dataset: Ionosphere. Average improvement is 1.63%.

Dataset	MH+Tree	MH+Forest	X+Forest
BreastCancerWisconsin	90.7	93.4	95.1
CarEvaluation	85.8	91.9	[89.2]
CreditApproval	91.4	91.5	93.1
GermanCredit	74.3	75.4	77.9
Ionosphere	87.1	[83.0]	92.5
MammographicMass	88.2	89.1	90.8
Mushroom	71.6	78.8	90.2
Nursery	92.9	93.2	94.0
SPECTHeart	93.4	93.7	94.9
TicTacToe	73.9	75.1	77.2
Vehicle	79.2	82.7	[81.9]
Wine	95.5	95.8	96.9

Table 6.3: Accuracy percents on the testing data (30% of $\mathcal{E}^{\succ}$) for UIUP trees and forests of 5000 UIUP trees, using the greedy and exact algorithms from the learning data (the other 70% of $\mathcal{E}^{\succ}$)

2. UIUP forests using the exact algorithm outperforms UIUP trees using the greedy method on all datasets. Average improvement is 4.14%.

3. UIUP forests using the exact algorithm outperforms UIUP forests using the greedy method on all but two datasets: CarEvaluation and Vehicle. Average improvement is 2.51%.

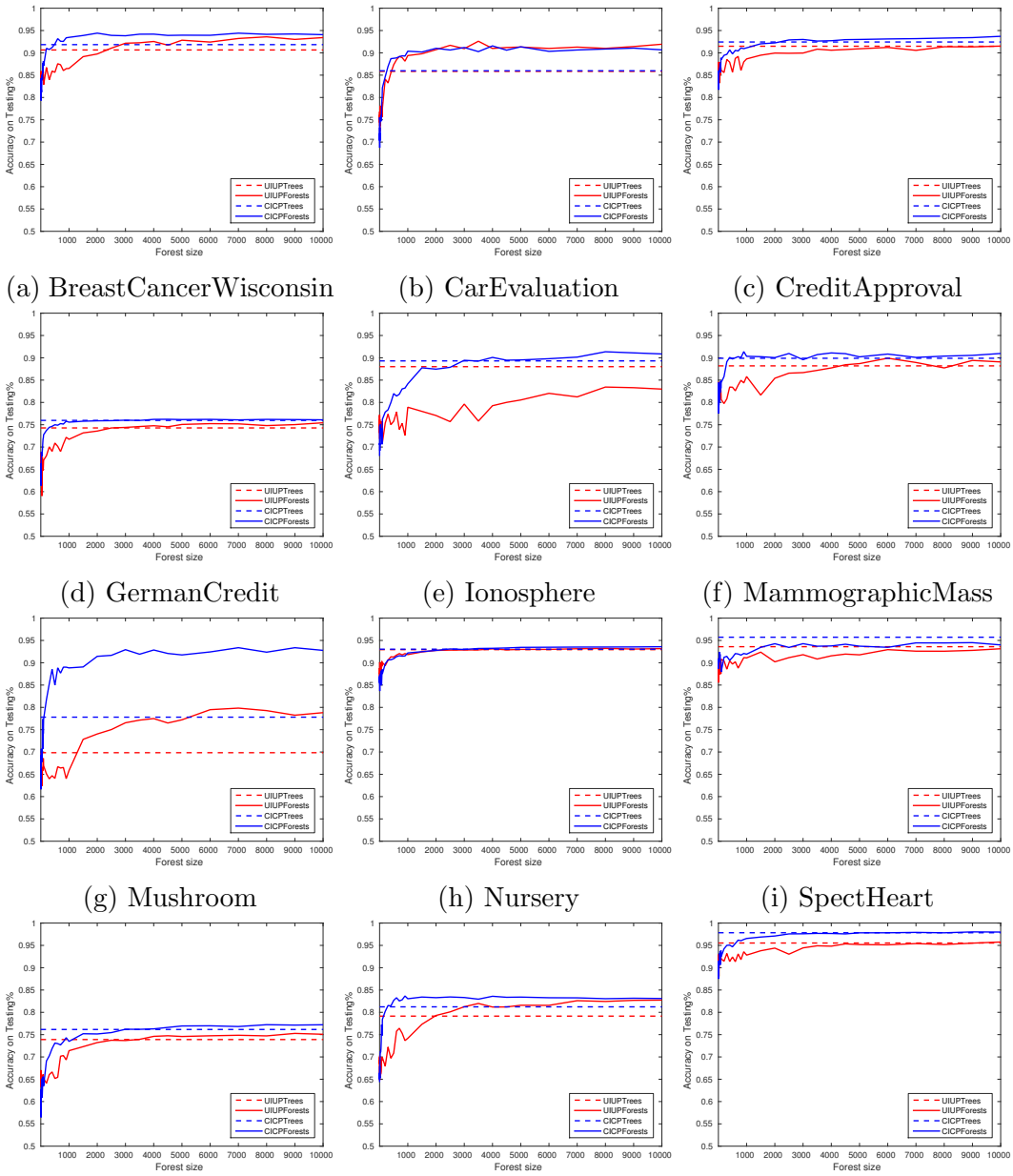
Second, we show results for UIUP PLP-forests using the greedy heuristic with different learning sample size per tree. The setting is similar to the previous one. These results are shown in Figure 6.6a, Figure 6.6b, Figure 6.6c, Figure 6.6e, Figure 6.6f, Figure 6.6i, Figure 6.6j, Figure 6.6k, and Figure 6.6l.

What we have learned from Figure 6.6:

1. CICP forests surpass CICP trees on all but one dataset: SPECTHeart.

2. UIUP forests surpass CICP trees on 4 datasets, and fall short on the others.

3. CICP forests dominate UIUP forests on 10 datasets, and perform very close on the other 2.



(a) BreastCancerWisconsin (b) CarEvaluation (c) CreditApproval

(d) GermanCredit (e) Ionosphere (f) MammographicMass

(g) Mushroom (h) Nursery (i) SpectHeart

(j) TicTacToe (k) Vehicle (l) Wine

Figure 6.6: Forests of UIUP trees vs. forests of CICP trees

6.7 Conclusion and Future Work

In this paper, we reviewed different classifications of PLP-trees, and introduced *partial lexicographic preference forests*, or *PLP-forests*, to reduce the high variance of PLP-trees. To support experimentation, we have constructed a preference learning library from existing reservoir of classification datasets. To this end, we implemented an

exact learner using *answer-set programming*, and an *approximation* learner using a straightforward greedy heuristic. Our empirical results on PLP-trees show that the language offers high accuracy, mostly higher than decision trees when training samples are small. For PLP-forests, we have observed improvements, significant in some cases, from single tree models.

Looking into the future, we are interested in expanding our preference learning library by creating real-work datasets through conducting experiments involving human subjects. We also plan to implement and experiment with other aggregators for PLP-forests, and compare with our results using the desirable and intuitive majority rule.

Chapter 7 Aggregating Lexicographic Preference Trees

Aggregating *votes* — preference orders over *candidates* or *alternatives* — is a fundamental problem of decision theory and social choice. We study this problem in the setting when alternatives are described as tuples of values of attributes. Such spaces of alternatives are called *combinatorial*. They are characterized by large sizes that make explicit enumerations of alternatives from the most to the least preferred infeasible. Instead, typically votes are specified implicitly in terms of some compact and intuitive preference representation mechanism. In our work, we assume that votes are given as *lexicographic preference trees* and consider two preference-aggregation problems, the *winner* problem and the *evaluation* problem. We study them under the assumption that *positional scoring rules* (such as k-approval and Borda) are used for aggregation. We develop computational complexity results for these two problems. We also propose computational methods to solve them. They are based on encodings of the problems in *Answer-Set Programming* and as instances of the *Weighted Partial Maximum Satisfiability* problem, and exploit off-the-shelf solvers available for these two formalisms. Finally, we present results of an experimental study of the effectiveness of these methods.

7.1 Introduction

Preferences are an essential component of decision making, social choice, knowledge representation, and constraint satisfaction. Fundamental problems of preference reasoning are to *aggregate* individual preference orders of a group of agents (the *votes* of agents in the group) into a consensus best candidate (the *winner*), and to identify candidates with strong consensus support from the group ("good" alternatives). These problems have been studied extensively in social choice [9]. Aggregation methods

known as *positional scoring rules*, which include such well-known rules as plurality, k-approval and Borda, are among the best understood and the most widely used ones.

When the number of alternatives is small, the simplest and most effective way to describe a preference order (a vote) is to enumerate the alternatives from the most to the least preferred. Moreover, given a collection of such votes, for many aggregation rules, including all positional scoring rules, computing winners and "good" candidates is easy — it can be done in polynomial time. The situation changes when alternatives are characterized in terms of *attributes* (or issues), and are specified by tuples of attribute values. Spaces of such alternatives, often called *combinatorial domains*, are large. Indeed, the number of alternatives grows exponentially with the number of attributes. This large size of combinatorial domains brings up two problems. First, it is no longer feasible to describe votes by enumerating alternatives in the order of preference. Thus, formalisms offering compact and intuitive representations of votes are needed. Several such *preference formalisms* have been developed over the years including penalty logic [27], possibilistic logic [30], conditional preference networks (CP nets) [17], preference trees [**?**], and lexicographic preference trees [**?**].[1] Second, when votes are given as expressions in some preference formalism, computing the winner or a "good" candidate is no longer easy. In fact, it is known that for many preference formalisms these problems are NP-hard even when positional scoring rules are used to aggregate votes. *Issue-by-attribute* aggregation addresses the computational hardness problem but often leads to results different from those obtained by applying common voting rules [32].

In this paper, we assume that votes are represented as *lexicographic preference trees*, or *LP-trees*, for short [16], and that they are aggregated by some simple positional scoring rules such as Borda, k-approval and a refinement of the latter, (k, l)-approval. Given this setting, we study computing the best alternative, and the related

[1] Kaci [49] offers a comprehnsive discussion of preference formalisms.

problem to decide whether an alternative with the score exceeding a given threshold (a "good" alternative) exists. We refer to the former problem as the *winner* problem and to the latter one as the *evaluation* problem. In our setting, these problems are often computationally hard. For Borda, the *winner* problem is NP-hard and the *evaluation* problem is NP-complete [53]. For k-approval, for some specific values of k, both problems are in P but, for some other, they are NP-hard and NP-complete, respectively [53]. Further, as we show in our paper, when (k, l)-approval is used, for several values of k and l, the problems are similarly hard.

Nevertheless, because the *winner* and the *evaluation* problems arise in practice and the positional scoring rules are common, computational tools for the two problems are needed. To develop such tools, we encode the problems in answer-set programming (ASP) [62, 65] and weighted partial maximum satisfiability (WPM-SAT) [6, 5], and apply to the encodings the ASP solvers *clingo* [41] and *clingcon* [66], and a WPM-SAT solver *toulbar*[4]. We chose the two ASP solvers as they represent substantially different approaches to computing answer sets. The *clingo* solver is a native ASP solver developed along the lines of satisfiability solvers. The *clingcon* solvers enhances *clingo* with specialized treatment of some common classes of numeric constraints by delegating some reasoning tasks to a CP solver *Gecode* [71]. As problems we are considering involve numeric constraints, a comparison of the two solvers is of interest. We study all the resulting methods experimentally. To support the experimentation we propose and implement a method to randomly generate LP-trees of some restricted form.

The main contributions of our work are complexity results and algorithms for the *winner* and the *evaluation* problems when votes are specified as LP-trees. Specifically, we present new complexity results for the two problems for several positional scoring rules: k-approval (for specific values of k), variants of Borda, and (k, l)-approval (for specific combinations of values of k and l). Next, we propose algorithms for the two

problems based on their ASP and WPM-SAT encodings and using ASP and WPM-SAT solvers. Finally, we provide an experimental evidence of the effectiveness of the proposed computational methods.

7.2 Computing Ranks

We now show how the *rank* of an outcome in an LP-tree is computed. As we consider positional scoring rules, the scores of an outcome in an LP-tree or an LP-profile under these rules follow directly from its rank in the tree.

Given an LP-tree T and an outcome $o \in CD(\mathcal{I})$, the computation of the rank $r(T, o)$ of o in T is given in Algorithm 3, where $T'(x_j)$ is the left (more-preferred) subtree of T', and $T'(\overline{x_j})$ is the right (less-preferred) subtree of T'. Note that in each case we need to update the CPT's in the subtrees accordingly. Clearly, Algorithm 3 takes $O(p)$. Conversely, it is also easy to compute the outcome at a given rank in a tree.

Algorithm 3: Compute the rank of an outcome in an LP-tree

Input: LP-tree T and outcome o
Output: the rank r of o in T

```
1  r←0;
2  T'←T;
3  for i←1 to p do
4  |  Let X_j be the root attribute of T' with preference x_j > x̄_j;
5  |  if o(X_j) = x_j then
6  |  |  T'←T'(x_j);
7  |  else
8  |  |  r←r + 2^{p−i};
9  |  |  T'←T'(x̄_j);
10 |  end
11 end
12 return r
```

Now computing the scores of an outcome for the rules k-approval, (k, l)-approval and Borda is straightforward. We have the following.

1. k-approval: $s_{kApp}(T, o) = 1$, if $r(T, o) < k$; 0, otherwise.

2. (k, l)-approval: $s_{klApp}(T, o) = a$, if $r(T, o) < k$; b, if $k \leq r(T, o) < k + l$; 0, otherwise.

3. Borda: $s_{Borda}(T, o) = m - r(T, o) - 1$.

7.3 The Problems and Their Complexity

We consider only *effective implicit* positional scoring rules, that is, rules defined by an algorithm that given m (the number of alternatives and, at the same time, the size of the scoring vector) and a rank r, $0 \leq r \leq m - 1$, (1) returns the value w_r of the scoring vector, and (2) works in time polynomial in the sizes of r and m. The rules k-approval, (k, l)-approval and Borda are examples of effective implicit positional scoring rules:

1. k-approval: $w_{kApp}(r, m) = 1$, if $r < k$; 0, otherwise.

2. (k, l)-approval: $w_{klApp}(r, m) = a$, if $r < k$; b, if $k \leq r < k + l$; 0, otherwise.

3. Borda: $w_{Borda}(r, m) = m - r - 1$.

Let us fix an effective implicit positional scoring rule $\mathcal{D}$ with the scoring vector w. Given an LP profile $\mathcal{V}$, the *winner* problem for $\mathcal{D}$ consists of computing an alternative $o \in \mathcal{X}$ with the maximum score $s_w(\mathcal{V}, o)$. Similarly, given a profile $\mathcal{V}$ and a positive integer R, the *evaluation* problem for $\mathcal{D}$ asks if there exists an alternative $o \in \mathcal{X}$ such that $s_w(\mathcal{V}, o) \geq R$. In each case, w is the scoring vector of $\mathcal{D}$ for m alternatives; we recall that it is given implicitly in term of an algorithm that efficiently computes its entries.

We apply the voting rules listed above to profiles consisting of LP-trees or *LP profiles*, for short. We distinguish four classes of profiles, UI-UP, UI-CP, CI-UP and CI-CP depending on the type of LP-trees they consist of.

Remark The restriction to effective implicit positional scoring rules is essential in the context of combinatorial domains. It is beacause an explicit specification of the scoring vector has size equal to the number of alternatives and is exponential in the number of attributes. If it were to be given explicitly, it would have to be a part of input. The sheer size of the scoring vector would then make both the winner and the evaluation problems trivially solvable in polynomial time. However, most interesting positional scoring rules are effective implicit, which means that they can be described concisely as an algorithm (implicit) and at the same time provide a fast access to any weight in the scoring vector (effective). In this setting, the complexity of the winner and the evaluation problems is no longer obvious, and it is precisely this setting that models practical situations, where scoring vectors are based on *regular* patterns.

k-Approval

If $k = 2^{p-1}$ the evaluation problem is in P for all four classes of profiles of LP-trees [53]. However, if k equals 2^{p-2} or 2^{p-3}, the problem is NP-complete, again for all four types of profiles [53] (in fact, the result holds for a larger set of values k, we refer for details to the paper by Lang et al. [53]). Clearly, in each case where the evaluation problem is NP-complete, the winner problem is NP-hard.

We first show that the two problems are in P even when the deviation of k from 2^{p-1} is given by a polynomial in p. In other words, if $k = 2^{p-1}+f(p)$ or $k = 2^{p-1}-f(p)$, where $f(p)$ is a polynomial in p such that $f(p) \geq 0$ for $p \geq 1$, both the winner and the evaluation problems for k-approval can be solved by polynomial time algorithms. The next two results address the two cases for k, respectively.

Theorem 17. *Let f be a polynomial such that $f(p) \geq 0$ for $p \geq 1$, and let $k = 2^{p-1} + f(p)$. Given a profile of n LP-trees over p binary attributes $X_1, \ldots, X_p$, the winner under k-approval can be computed in time polynomial in the size of the profile.*

Proof. Let P be a profile of n LP-trees. The score $s_k(o)$ of an alternative o under k-approval in P is given by

$$s_k(o) = s'(o) + s''(o),$$

where $s'(o)$ is the score of o under the 2^{p-1}-approval (the number of votes that place o in the upper half of the order), and $s''(o)$ is the number of votes that place o as one of top $f(p)$ votes in the lower half (we omit references to the profile to simplify the notation).

To find the highest possible score $s'(o)$, we define $x_i = 0$, if the number of votes with the root labeled with X_i and with 0 preferred to 1 is *strictly* larger than the number of votes with the root labeled with X_i with 1 preferred to 0. We define $x_i = 1$ similarly. If x_i does not get set to 0 or 1, it is set to u (undefined). We call the resulting p-tuple a *partial alternative* and denote it by PA. Since it is the root that decides whether an LP-tree contributes 1 to the score of an alternative, it is clear that any alternative consistent with PA achieves the highest possible score under 2^{p-1}-approval, that is, the highest possible s'-score. Finding this score, say W', can then be accomplished by (1) finding an alternative o consistent with PA, and (2) finding its score $s'(o)$. Clearly, both (1) and (2) together can be done in time bounded by a polynomial in the size of the profile.

Next, we consider s''. Let us denote by A the set of all alternatives o with $s''(o) > 0$. To this end, it is enough to find in each tree T in P alternatives with ranks $2^{p-1}+1, \ldots, 2^{p-1}+f(p)$. Since finding an alternative of a given rank in an LP-tree can be accomplished in time polynomial in p, the set A can indeed be computed in time polynomial in the size of the profile.

We now compute $s_k(o)$ for all alternatives in A. Given the size of A, the task can be computed in time bounded by a polynomial in the size of the profile. Let W be

the maximum of these scores achieved, say, by an alternative o. If $W \geq W'$, then o is a winning alternative (has the best score among those in A and the score of any other alternative does not exceed W'). Otherwise, any alternative consistent with PA can be taken for the winner (indeed, in such case, the highest possible score to achieve under k-approval is W'). □

Theorem 18. *Let f be a polynomial such that $f(p) \geq 0$ for $p \geq 1$, and let $k = 2^{p-1} - f(p)$. Given a profile of n LP-trees over p binary attributes $X_1, \ldots, X_p$, the winner under k-approval can be computed in time polynomial in the size of the profile.*

Proof. Let P be a profile of n LP-trees. Similarly as in the proof of the previous result, the score $s_k(o)$ of an alternative o under k-approval is given by

$$s_k(o) = s'(o) - s''(o),$$

where $s'(o)$ is the score of o under the 2^{p-1}-approval (the number of votes that place o in the upper half of the order), and $s''(o)$ is the number of votes that place o as one of the bottom $f(p)$ votes in the upper half (we omit references to the profile to simplify the notation).

Let us denote by A the set of alternatives o such that $s''(o) > 0$. As before, this set can be computed in time bounded by a polynomial in the size of the profile. Let $t = |A|$. If every alternative is in A (that is, $t = 2^p$), then, we compute an alternative with the highest k-approval score by computing the scores of all alternatives in A and selcting the one with the highest score. Since the size of A is polynomial in the size of the profile, the task takes polynomial time (in the size of the profile).

The case when $t < 2^p$ is harder. To address it, let us assume that we have computed the set B of top $t + 1$ alternatives according to their s'-score (the 2^{p-1}-approval score). Next, let o be an alternative in B with the maximum k-approval score $s_k(o)$.

We claim that o is also an alternative with the maximum k-approval score over all alternatives. Indeed, consider an arbitrary alternative o'. If $o' \in B$, then $s_k(o) \geq s_k(o)$ (by the way o was selected). Thus, let us assume that $o' \notin B$. Since $|B| > |A|$, there is at least one alternative $o'' \in B \setminus A$. As $o'' \in B$, $s_k(o) \geq s_k(o'')$. Morever, as $o'' \notin A$, $s_k(o'') = s'(o'') - s''(o'') = s'(o'')$. Finally, since $o'' \in B$ and $o' \notin B$, $s'(o'') \geq s'(o')$. Combining these three inequalities, we obtain that $s_k(o) \geq s'(o')$. Since $s'(o') \geq s'(o') - s'(o'') = s_k(o')$, we get $s_k(o) \geq s_k(o')$. Thus, the claim follows.

Clearly, $t + 1$ is bounded by a polynomial in the size of the profile. Thus, once B is computed, finding an alternative in B with the highest k-approval score can be done in time polynomial in the size of the profile. To complete the proof, it suffices then to show how to compute B in polynomial time.

To this end, for each $i = 1, \ldots, p$, we set d_i to the absolute value of the difference between the numbers of trees in the profile with the root labeled with X_i and with 0 (respectively, with 1) as the preferred value. We also select any alternative that has the highest s'-score (we explained in the previous proof how to compute it in polynomial time) and denote it by o. Finally, we compute the score of o and denote it by W' (to use the notation from the previous proof).

Let $S \subseteq \{1, \ldots, p\}$ be a set of attribute indices, and let o_S be an alternative obtained from o by "flipping" its values in positions in S. Every alternative can be described in these terms. This is useful as the s'-score of o_S is easy to compute. Namely, we have

$$s'(o_S) = W' - w(S),$$

where $w(S) = \sum_{i \in S} d_i$ is the *weight* of S.

It follows that B is determined by $t+1$ smallest-weight subsets of $\{1, \ldots, p\}$. We will now show that given a list $D = \{d_1, d_2, \ldots, d_p\}$ and an integer t, the $t+1$ smallest-weight subsets of $\{1, \ldots, p\}$ can be computed in time bounded by a polynomial in p and t.

Let r be an integer such that $2^r \geq t+1$. Let us assume that L_r is the set of $t+1$ smallest-weight subsets of $\{1,\ldots,r\}$. Let

$$L'_{r+1} = L_r \cup \{S \cup \{r+1\} \colon S \in L_r\}$$

and let L_{r+1} be the collection of $t+1$ smallest-weight subsets S of L'_{r+1}. We will show that L_{r+1} contains $t+1$ smallest-weight subsets S of $\{1,\ldots,r+1\}$. Indeed, let us consider $S \subseteq \{1,\ldots,r+1\}$ such that $S \notin L'_{r+1}$. If $S \subseteq \{1,\ldots,r\}$, then $S \notin L_r$. Thus, $w(S) \geq w(S')$, for every $S' \in L_r$. If $r+1 \in S$, then $S = R \cup \{r+1\}$, for some $R \subseteq \{1,\ldots,r\}$. Since $S \notin L'_{r+1}$, $R \notin L_r$. Thus, $w(R) \geq w(R')$, for every $R' \in L_r$ and so, $w(S) \geq w(R' \cup \{r+1\})$ for all $R' \in L_r$. In each case, it follows that there are at least $t+1$ sets S' in L'_{r+1} such that $w(S) \geq w(S')$. Thus for every $S' \in L_{r+1}$, $w(S) \geq w(S')$.

Clearly, the list L_p consists of $t+1$ smallest weight subsets of $\{1,\ldots,p\}$. Thus, it can be taken for B. To compute it, we first find the smallest r such that $2^r \geq t+1$ (such an r exists as we are now considering the case when $t < 2^p$). We then construct the collection U of all subsets of $\{1,\ldots,r\}$ (this collection has no more than $2t$ elements and can be constructed in time bounded by a polynomial in p and t). Next, we construct L_r by selecting from U its $t+1$ smallest-weight elements. Since $|U| \leq 2t$, this task also can be accomplish in polynomila time (in p and t).

From now on, we construct $L_{r+1}, L_{r+2}, \ldots L_p$ recursively, as described above. Since each step of the construction can be accomplished by the same polynomial-time algorithm (form the collection L', select its $t+1$ smallest-weight elements to form the next L), and since the number of steps is boundend by p, the total time needed to construct B (L_p) is boundend by a polynomial in p and t. □

For the k-approval rule, we summarize the results in Table 7.1, where Table 7.1a presents our results as discussed above, and results in Table 7.1b were obtained by

	UP	CP
UI	P	P
CI	P	P

(a) $k = 2^{p-1} \pm f(p)$

	UP	CP
UI	NPC	NPC
CI	NPC	NPC

(b) $k = c \cdot 2^{p-M}$ and $k \neq 2^{p-1}$

Table 7.1: k-Approval

others [53].

(k, l)-Approval

To the best of our knowledge, the complexity of the 2-valued (k, l)-approval rule has not been studied. It is evident that (k, l)-approval is an effective implicit positional scoring rule. It turns out that, as with the k-approval rule, for some values of the parameters, the evaluation problem for (k, l)-approval is NP-complete. Preliminary results we obtained have been published [56]. We describe cases where $k = l = 2^{p-c}$, where c is a constant and $1 < c < p$. If $a = 2$ and $b = 1$, we refer to the rule $(2^{p-2}, 2^{p-2})$-approval as $2K$-*approval*. We show proof of NP-completeness of the evaluation problem for $(2^{p-2}, 2^{p-2})$-approval.

Theorem 19. *The following problem is NP-complete: decide for a given UI-UP profile $\mathcal{V}$ and an integer R whether there is an alternative o such that $s_w(\mathcal{V}, o) \geq R$, where w is the scoring vector of the $(2^{p-2}, 2^{p-2})$-approval rule.*

Proof. We can guess in polynomial time an alternative $o \in \mathcal{X}$ and verify in polynomial time that $S_w(\mathcal{V}, o) \geq R$ (this is possible because (k, l)-approval is an effective implicit scoring rule; the score of an alternative in a vote can be computed in polynomial time once its position is known, and the position can be computed in polynomial time be traversing the tree representing the vote). So membership in NP follows. Hardness follows from a polynomial reduction from the problem 2-*MINSAT*[2] [50], which is

[2]Let N be an integer ($N > 1$), the N-MINSAT problem is defined as follows. Given a set Φ of n N-clauses $\{c_1, \ldots, c_n\}$ over a set of propositional variables $\{X_1, \ldots, X_p\}$, and a positive integer l ($l \leq n$), decide whether there is a truth assignment that satisfies at most l clauses in Φ.

NP-complete. Given an instance $\langle \Phi, l \rangle$ of the 2-MINSAT problem, we construct the set of attributes $\mathcal{I}$, the set of alternatives $\mathcal{X}$, the profile $\mathcal{V}$ and the threshold R.

Important observations are that o is among the top first quarter of alternatives in an LP-tree $\mathcal{L}$ if and only if the top two most important attributes in $\mathcal{L}$ are both assigned the preferred values; and that o is among the second top quarter of alternatives if and only if the most important attribute is assigned the preferred value and the second most important one is assigned the non-preferred one.

(1). We define $\mathcal{I} = \{X_1, \ldots, X_p\}$, where X_is are all propositional letters occurring in Φ. Clearly, the set $\mathcal{X}$ of all alternatives over $\mathcal{I}$ coincides with the set of truth assignments of variables in $\mathcal{I}$.

(2). Let Ψ be the set of formulas $\{\neg c_i : c_i \in \Phi\}$. For each $\neg c_i \in \Psi$, we build $a + b$ UI-UP trees. For instance, if $\neg c_i = X_2 \wedge \neg X_4$, then we proceed as follows. Firstly, we build $a - b$ duplicate trees shown in Figure 7.1a. Secondly, we construct b duplicate trees shown in Figure 7.1b. Thirdly, we build another b duplicate trees shown in Figure 7.1c. (In all three figures we only indicate the top two attributes since the other attributes can be ordered arbitrarily.) Denote by $\mathcal{V}_i$ the set of these $a+b$ UI-UP trees for formula $\neg c_i$. Then $\mathcal{V} = \bigcup_{1 \leq i \leq n} \mathcal{V}_i$ and has $n * (a + b)$ votes.

(3). Finally, we set $R = (n - l) * (a^2 - ab + b^2) + l * ab$.

Note that the construction of $\mathcal{V}$ ensures that if $o \models \neg c_i$, $S_w(\mathcal{V}_i, o) = a^2 - ab + b^2$; otherwise if $o \not\models \neg c_i$, $S_w(\mathcal{V}_i, o) = ab$. We have $a^2 - ab + b^2 > ab$ since $(a - b)^2 > 0$. Hence, there is an assignment satisfying at most l clauses in Φ if and only if there is an assignment satisfying at least $n - l$ formulas in Ψ if and only if there is an alternative with the $(2^{p-2}, 2^{p-2})$-approval score of at least R given the profile $\mathcal{V}$.

Since the first equivalence is clear, it suffices to show the second. Let o be an assignment satisfying l' formulas in Ψ. We have $S_w(\mathcal{V}, o) - R = (l' + l - n) * (a^2 - ab + b^2) + (n - l' - l) * ab = (l' + l - n) * (a^2 - 2ab + b^2) = (l' + l - n) * (a - b)^2$. It follows that $S_w(\mathcal{V}, o) \geq R$ if and only if $l' + l - n \geq 0$ if and only if $l' \geq n - l$. $\square$

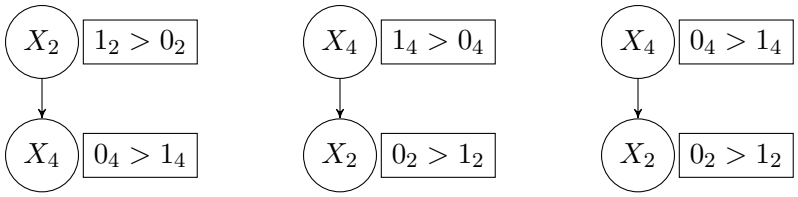


Figure 7.1: UI-UP LP-trees

This hardness proof applies to more general classes of LP-trees, namely UI-CP, CI-UP and CI-CP, and the winner problem for those cases is NP-hard. Below we show the proof of NP-completeness of the evaluation problem for $(2^{p-3}, 2^{p-3})$-approval.

Theorem 20. *Let w be the scoring vector $(a, \ldots, a, b, \ldots, b, 0 \ldots, 0)$ with the numbers of a's and b's each equal to 2^{p-3}. The problem of deciding for a given UI-UP profile V and an integer R whether there is an alternative o such that $s_w(V, o) \geq R$ is NP-complete.*

Proof. We can guess in polynomial time an alternative $o \in \mathcal{X}$ and verify in polynomial time that $S_w(V, o) \geq R$. So membership in NP follows.

Hardness follows from a polynomial reduction from the NP-complete problem 3-*MAXSAT* [68]. Let Φ be a set of n 3-clauses $\{c_1, \ldots, c_n\}$ over $\{X_1, \ldots, X_p\}$, l an integer such that $0 \leq l \leq n$. Given an instance of 3-*MAXSAT* $I = \langle \Phi, l \rangle$, we construct the set of attributes X, the set of alternatives $\mathcal{X}$, the profile V and the threshold R as follows.

(1) $X = \{X_1, \ldots, X_p\}$. $\mathcal{X}$ is then the set of all alternatives over X.

(2) Let Ψ be the set of formulas $\{\neg c_i : c_i \in \Phi\}$. For each $\neg c_i \in \Psi$, we build multiple UI-UP LP-trees. Assume there is $c_i = \neg X_1 \vee \neg X_2 \vee \neg X_3 \in \Phi$. Then we have $\neg c_i = X_1 \wedge X_2 \wedge X_3 \in \Psi$. For $\neg c_i$, we build a^2 duplicate trees of type 7.2a, a^2 duplicate trees of type 7.2b, a^2 duplicate trees of type 7.2c, a^2 duplicate trees of type 7.2d, $a^2 - ab$ duplicate trees of type 7.2e, $a^2 - ab$ duplicate trees of type 7.2f and

	UP	CP
UI	P	P
CI	P	P

(a) $k = l = 2^{p-1}$

	UP	CP
UI	NPC	NPC
CI	NPC	NPC

(b) $k = l = 2^{p-c}$ and $1 < c < p$

Table 7.2: (k, l)-Approval

$(a-b)^2$ duplicate trees of type 7.2g. Denote by V_i the set of $7a^2 - 4ab + b^2$ UI-UP LP-trees for formula $\neg c_i$. Then $V = \bigcup_{1 \leq i \leq n} V_i$ and has $n * (7a^2 - 4ab + b^2)$ votes.

(3) We set $R = a^3 * l + (3a^2b - 3ab^2 + b^3) * (n - l)$.

Note that the construction of V ensures that if $o \models \neg c_i$, $S_w(V_i, o) = 3a^2b - 3ab^2 + b^3$; otherwise if $o \not\models \neg c_i$, $S_w(V_i, o) = a^3$. We have $a^3 > 3a^2b - 3ab^2 + b^3$ since $a^3 - (3a^2b - 3ab^2 + b^3) = (a-b)^3 > 0$. Therefore, there is an assignment satisfying at least l clauses in Φ *iff* there is an assignment falsifying at least l formulas in Ψ *iff* there is an alternative scoring at least R with respect to profile V and our scoring vector w. Since the first equivalence is obvious, it suffices to show the second one.

($\Rightarrow$) Assume o is the assignment that falsifies l' ($l' \geq l$) formulas in Ψ, its score $S_w(V, o) = a^3 * l' + (3a^2b - 3ab^2 + b^3) * (n - l')$. Then $S_w(V, o) - R = a^3 * (l' - l) + (3a^2b - 3ab^2 + b^3) * (l - l') = a^3 * (l' - l) - (3a^2b - 3ab^2 + b^3) * (l' - l) = (a-b)^3 * (l' - l) \geq 0$. Thus, $S_w(V, o) \geq R$.

($\Leftarrow$) Suppose o is the alternative such that $S_w(V, o) \geq R$. Prove by contradiction. Assume o falsifies l' formulas in Ψ and $l' < l$. Then $S_w(V, o) - R = (a-b)^3 * (l' - l) < 0$, which implies that $S_w(V, o) < R$. Contradiction! Therefore, it must be that $l' \geq l$.

□

For the (k, l)-approval rule, we capture our results as Table 7.2.

b-Borda

By *b-Borda* we mean a positional scoring rule with the scoring vector $\langle b, b-1, b-2, \ldots \rangle$. Let $m = 2^p$ denote the number of alternatives in $\mathcal{X}(\mathcal{I})$ (where, as always, $\mathcal{I} =$

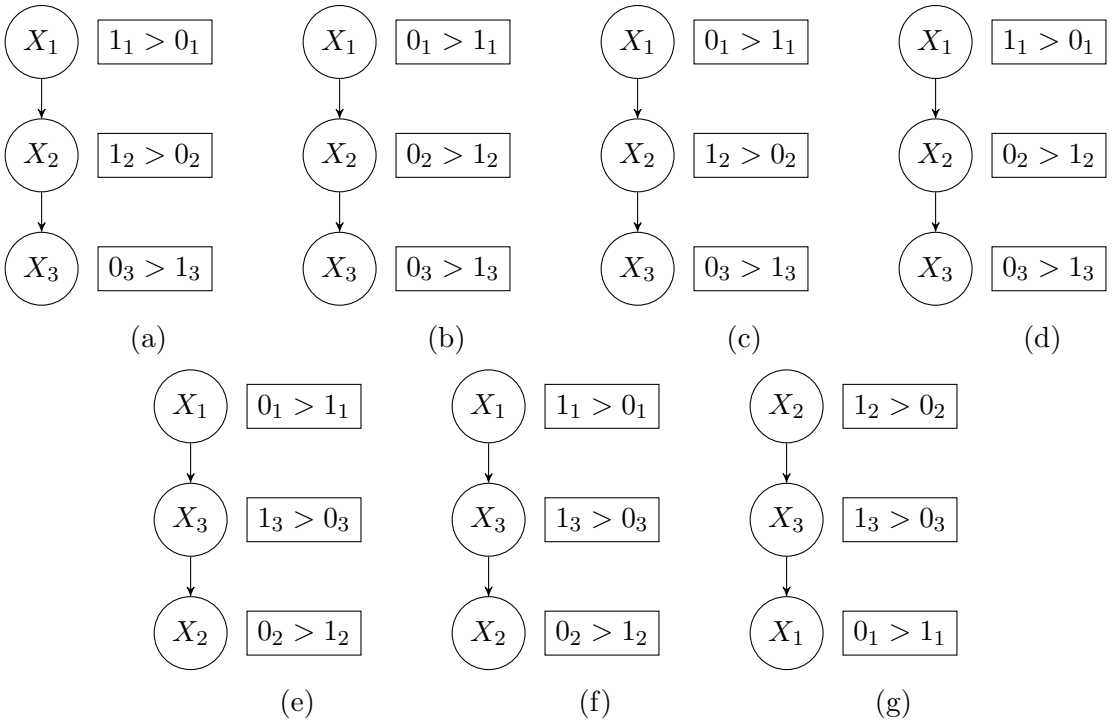


Figure 7.2: UI-UP LP-trees

$\{X_1, \ldots, X_p\}$). If $b \geq 2^p - 1$, b-Borda can be reduced to the (standard) Borda rule. In the most restrictive case of UI-UP profiles, the evaluation problem for the Borda rule is in P, and it is NP-complete for the three other classes of profiles [53].

When $b < 2^p - 1$, we show that for some values of b, the winner and the evaluation problems under the b-Borda rules are NP-hard and NP-complete, respectively, no matter what the type of LP-trees used in profiles. The cases of UI-CP, CI-UP and CI-CP trees are handled by a fairly direct reduction from the corresponding problems under the Borda rule. The case of UI-UP profiles requires a different argument (the winner and the evaluation problems under the standard Borda rule are, as we noted, in P). We start with the latter.

We denote by *half-Borda* the b-Borda rule with $b = 2^{p-1} - 1$, where p is the number of attributes in $\mathcal{I}$. We have the following theorem on half-Borda.

Theorem 21. *The evaluation and the winner problems under half-Borda for UIUP-profiles are NP-complete and NP-hard, respectively.*

Proof. We show that the evaluation problem is NP-complete. The membership in NP is obvious. The NP-hardness follows from a polynomial reduction from the 2-MINSAT problem.

Given a 2-MINSAT instance (Φ, l), where Φ consists of 2-clauses $C_1, \ldots, C_m$ over variables $X_1, \ldots, X_p$, we construct an instance of our problem as follows.

First, we introduce a new binary variable X_q and define the set of attributes $\mathcal{I}$ by setting $\mathcal{I} = \{X_1, \ldots, X_p, X_q\}$.

Second, for each $C_i \in \Phi$, we now build a set P_i of 12 UI-UP LP-trees over $\mathcal{I}$. As an example, let C_i be $\neg X_2 \vee X_4$[3]. The fragment of the profile determined by C_i is given by the multi-set

$$P_i = \{B_{i_1}, B_{i_2}, B_{i_1}, B_{i_2}, B_{i_1}, B_{i_2}, B'_{i_1}, B'_{i_2}, B''_{i_1}, B''_{i_2}, B''_{i_1}, B''_{i_2}\},$$

where the trees $B_{i_1}, B_{i_2}, B'_{i_1}, B'_{i_2}, B''_{i_1}$, and B''_{i_2} are shown in Figure 7.3. In other words, the profile P_i contains three copies of B_{i_1} and B_{i_2}, one copy of B'_{i_1} and B'_{i_2}, and two copies of B''_{i_1} and B''_{i_2}. We define the overall profile P as the collection of all profiles P_i, $1 \leq i \leq m$. That is, $P = \bigcup_{1 \leq i \leq m} P_i$. Clearly, we have $12 \cdot m$ UI-UP LP-trees in the profile P.

Finally, we set the threshold value $R = 15a \cdot (m - l) + 3a \cdot l$, where we use a to denote 2^{p-1}.

Let o be an outcome over $\mathcal{I}$. Let B be a UIUP tree over $\mathcal{I}$, X_j the most important attribute of B. We define the half-borda score of o in tree B, denoted by $s_{HB}(B, o)$, to be 0 if outcome o has the non-preferred value on X_j; $s_{Borda}(B_{|_{\mathcal{I}\setminus\{X_j\}}}, o_{|_{\mathcal{I}\setminus\{X_j\}}})$, otherwise. We now compute the half-Borda score of o according to whether it satisfies

[3]We will build P_i according to what C_i contains: the two atoms in C_i are the labels of the top two levels of trees, and whether the atom is negated affects the preference on that atom.

X_q and C_i. If $o \models X_q \wedge \neg C_i$, that is, $o \models X_q \wedge X_2 \wedge \neg X_4$, we have

$$s_{HB}(P_i, o) = \underbrace{(2^p - 1 + 2^{p-1} + 1) * 3}_{\text{three copys of } B_{i_1} \text{ and } B_{i_2}} + \underbrace{(0)}_{B'_{i_1} \text{ and } B'_{i_2}} + \underbrace{(2^p - 1 + 2^{p-1} + 1) * 2}_{\text{two copys of } B''_{i_1} \text{ and } B''_{i_2}}$$
$$= 15a.$$

If $o \models X_q \wedge C_i$, we need to consider three cases:

(1). If $o \models X_q \wedge \neg X_2 \wedge X_4$, we have

$$s_{HB}(P_i, o) = \underbrace{(0) * 3}_{\text{three copys of } B_{i_1} \text{ and } B_{i_2}} + \underbrace{(2^p - 1 + 2^{p-1} + 1)}_{B'_{i_1} \text{ and } B'_{i_2}} + \underbrace{(0) * 2}_{\text{two copys of } B''_{i_1} \text{ and } B''_{i_2}}$$
$$= 3a.$$

(2). If $o \models X_q \wedge \neg X_2 \wedge \neg X_4$, we have

$$s_{HB}(P_i, o) = \underbrace{(0) * 3}_{\text{three copys of } B_{i_1} \text{ and } B_{i_2}} + \underbrace{(2^{p-1} - 1 + 1)}_{B'_{i_1} \text{ and } B'_{i_2}} + \underbrace{(2^{p-1} - 1 + 1) * 2}_{\text{two copys of } B''_{i_1} \text{ and } B''_{i_2}}$$
$$= 3a.$$

(3). If $o \models X_q \wedge X_2 \wedge X_4$, we have

$$s_{HB}(P_i, o) = \underbrace{(2^{p-1} - 1 + 1) * 3}_{\text{three copys of } B_{i_1} \text{ and } B_{i_2}} + \underbrace{(0)}_{B'_{i_1} \text{ and } B'_{i_2}} + \underbrace{(0)}_{\text{two copys of } B''_{i_1} \text{ and } B''_{i_2}}$$
$$= 3a.$$

Thus, for $o \models X_q \wedge C_i$, we have $s_{HB}(P_i, o) = 3a$.

Similary, we can compute that $s_{HB}(P_i, o) < 15a$, if $o \models \neg X_q \wedge \neg C_i$; and $s_{HB}(P_i, o) < 3a$, if $o \models \neg X_q \wedge C_i$.

We now show that there exists an outcome over $\mathcal{I}$ with score at least R if and

only if there exists an assignment over I that satisfies at most l clauses in Φ.

($\Leftarrow$) We assume there is an assignment v over I satisfying at most l clauses in Φ. Define an outcome $o = (v, 1_q)$. It is clear that $s_{HB}(P, o) \geq R$.

($\Rightarrow$) We assume there is an outcome o over $\mathcal{I}$ such that $s_{HB}(P, o) \geq R$. If $o \models \neg X_q$, we could flip the value on X_q from 0_q to 1_q, and obtain o' such that $s_{HB}(P, o') > s_{HB}(P, o) \geq R$. Assuming $o'|_I$ satisfies l' ($l' > l$) clauses in Φ, we have that $s_{HB}(P, o') = 15a \cdot (m - l') + 3a \cdot l' > R$; thus, $l' < l$. A contradiction! Otherwise, if $o \models X_q$, we are done. □

Corollary 22. *Theorem Theorem 21 holds for b-Borda when $b = 2^{p-c} - 1$, where c is a constant and $1 \leq c < p$.*

Corollary 22 holds because we can construct c to be 1 and then the proof of Theorem 21 follows.

Theorem 23. *Let $b = 2^{p-c} - 1$, where p is the number of attributes and c a fixed integer such that $1 \leq c < p$. The evaluation and the winner problems under b-Borda for profiles consisting of CI-UP trees (UI-CP and CI-CP trees, respectively) are NP-complete and NP-hard, respectively.*

Proof. We only show an argument for the class CI-UP. The reasoning for other two types of profiles is similar. Moreover, we only show that the evaluation problem (under the restriction to profiles consisting of CI-UP trees) is NP-complete. Indeed, it directly implies that the corresponding variant of the winner problem is NP-hard.

As in other arguments befre, the membership in the class NP is evident. Thus, we focus on the hardness part of the argument. To show NP-hardness, we construct a reduction from the evaluation problem under Borda when profiles consist of CI-UP trees ($Borda^{ev}_{CI-UP}$, for short). That problem is known to be NP-complete [53].

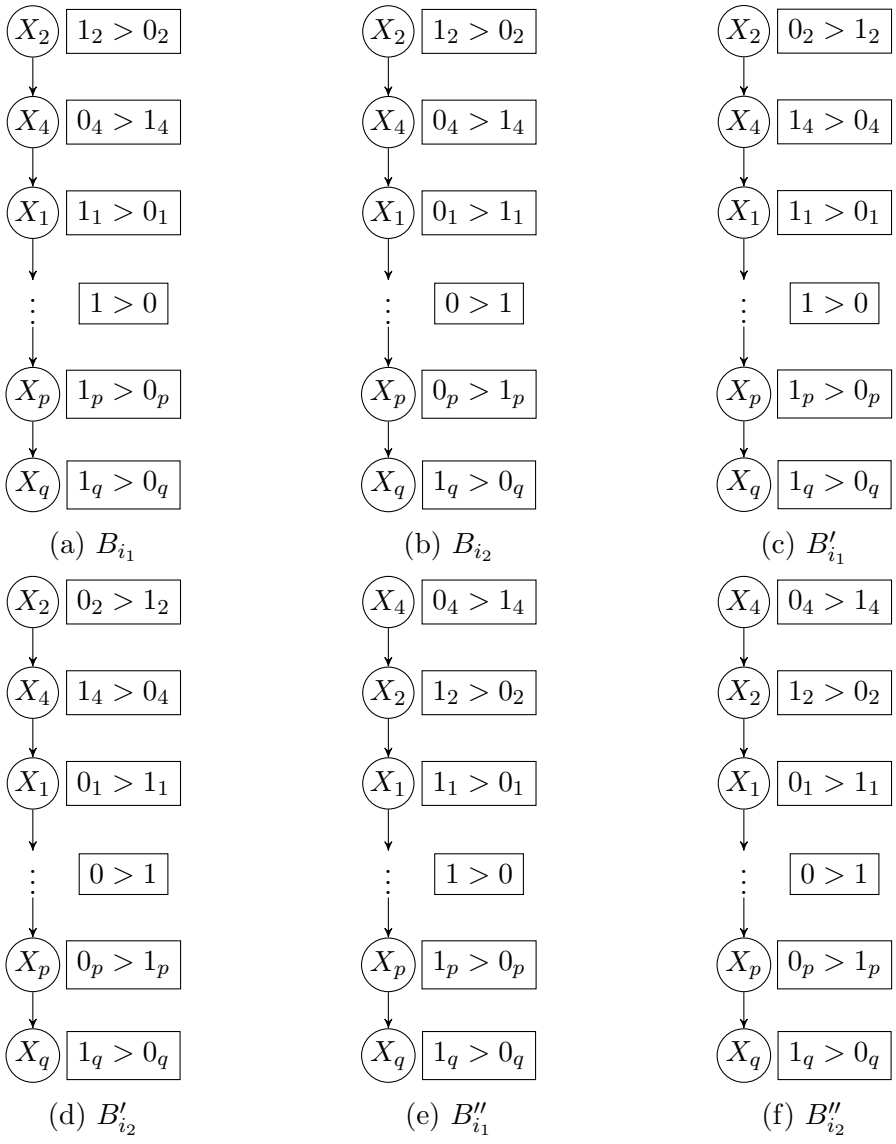


Figure 7.3: UI-UP LP-trees

Given an instance $\langle I, P, l\rangle$ of $Borda^{ev}_{CI-UP}$, where I is a set of p attributes $X_1, \ldots, X_p$, $P = \langle T_1, \ldots, T_m$ is a profile of m CI-UP trees over I, and l is a positive integer, we construct an instance $\langle \mathcal{I}, \mathcal{P}, \ell\rangle$ of our problem as follows.

First, we define $\mathcal{I} = \{Y_1, \ldots, Y_c, X_1, \ldots, X_p\}$, where $Y_1, \ldots, Y_c$ are new attributes. Second, we construct a UI-UP tree T built of c nodes labeled $Y_1, \ldots Y_c$ (from top to bottom), with the node labeled with Y_i having a local preference $1 > 0$. Then, for each $T_i \in P$, $1 \leq i \leq m$, we form a CI-UP tree T'_i by connecting the bottom node

	UP	CP
UI	P	NPC
CI	NPC	NPC

(a) $b = 2^p - 1$

	UP	CP
UI	NPC	NPC
CI	NPC	NPC

(b) $b = 2^{p-c} - 1$ and $1 \leq c < p$

Table 7.3: b-Borda

of T (the one labeled wit Y_c) by a "straight-down" edge to the root of T_i. We define $\mathcal{V} = \{T'_1, \ldots, T'_,\}$. Finally, we set $\ell = l$.

It is simple to verify that under the profile P there is an alternative with the Borda score of at least l if and only if under the profile $\mathcal{P}$ there is an alternative with the b-Borda score of at least ℓ. □

For the b-Borda rule, we include the complexity results in Table 7.3, where Table 7.3b shows existing results by others [53], and Table 7.3a presents results obtained by us.

7.4 The Problems in Answer-Set Programming

The winner and the evaluation problems are in general intractable in the setting we consider. Yet, they arise in practice and computational tools to handle them are needed. We develop and evaluate a computational approach based on *answer-set programming* (ASP) [62]. We propose several ASP encodings for both problems for the Borda, k-approval, and (k, l)-approval rules (for the lack of space only the encodings for Borda are discussed). The encodings are adjusted to two ASP solvers for experiments: *clingo* [41], and *clingcon* [66] and demonstrate the effectiveness of ASP in modeling problems related to preference aggregation.

Encoding LP Trees As Logic Programs

In the winner and evaluation problems, we use LP-trees only to compute the ranking of an alternative. Therefore, we encode trees as program rules in a way that enables

that computation for a given alternative. In the encoding, an alternative o is represented by a set of ground atoms $eval(i, x_i)$, $i = 1, 2, \ldots, p$ and $x_i \in \{0, 1\}$. An atom $eval(i, x_i)$ holds precisely when the alternative o has value x_i on attribute X_i.

If X_i is the attribute labeling a node t in vote v at depth d_i^v, $CPT(t)$ determines which of the values 0_i and 1_i is preferred there. Let us assume $\mathcal{P}(t) = \{t_1, \ldots, t_j\}$ and $Inst(t) = \{t_{j+1}, \ldots, t_\ell\}$, where each t_q is labeled by X_{i_q}. The location of t is determined by its depth d_i^v and by the set of values $x_{i_{j+1}}, \ldots, x_{i_\ell}$ of the attributes labeling $Inst(t)$ (they determine whether we descend to the left or to the right child as we descend down the tree). Thus, $CPT(t)$ can be represented by program rules as follows. For each row $u : 1_i > 0_i$ in $CPT(t)$, where $u = x_{i_1}, \ldots, x_{i_j}$, we include in the program the rule

$$\begin{aligned} vote(v, d_i^v, i, 1) \text{ :- } & eval(i_1, x_{i_1}), \ldots, eval(i_j, x_{i_j}), \\ & eval(i_{j+1}, x_{i_{j+1}}), \ldots, eval(i_\ell, x_{i_\ell}) \end{aligned} \tag{7.1}$$

(and similarly, in the case when that row has the form $u : 0_i > 1_i$).

In this representation, the property $vote(v, d_i^v, i, a_i)$ will hold true for an alternative o represented by ground atoms $eval(i, x_i)$ *precisely when* (or *if*, denoted by ":-" in our encodings) that alternative takes us to a node in v at depth d_i^v labeled with the attribute X_i, for which at that node the value a_i is preferred. Since, in order to compute the score of an alternative on a tree v all we need to know is whether $vote(v, d_i^v, i, a_i)$ holds (cf. our discussion below), this representation of trees is sufficient for our purpose.

For example, the LP-tree v in Figure 3.3 is translated into the logic program in Figure 7.4 (*voteID(v)* identifies the id of the vote (LP-tree)).

```
1  voteID(1).
2  vote(1,1,1,1).
3  vote(1,2,2,1) :- eval(1,1).
4  vote(1,3,3,1) :- eval(2,1), eval(1,1).
5  vote(1,3,3,0) :- eval(2,0), eval(1,1).
6  vote(1,2,3,0) :- eval(1,0).
7  vote(1,3,2,0) :- eval(1,0).
```

Figure 7.4: Translation of v in logic rules

Encoding Positional Scoring Rules In ASP

Encoding the Borda evaluation problem in *clingo*

The evaluation and the winner problems for Borda can be encoded in terms of rules on top of those that represent an LP profile. Given a representation of an alternative and of the profile, the rules evaluate the score of the alternative and maximize it or test if it meets or exceeds the threshold.

We first show the encoding of the Borda evaluation problem in *clingo* (Figure 7.5). Parameters in the evaluation problem are defined as facts (lines 1-4): predicates *at-*

```
1  attribute(1). attribute(2). attribute(3).
2  numIss(3).
3  val(0). val(1).
4  threshold(5).
5  1{ eval(I,M) : val(M) }1 :- attribute(I).
6  wform(V,I,W) :- vote(V,D,I,A), eval(I,A), numIss(P), W=#pow(2,P-D).
7  wform(V,I,0) :- vote(V,D,I,A), eval(I,M), A != M.
8  goal :- S = #sum [ wform(V,I,W) = W ], threshold(TH), S >= TH.
9  :- not goal.
```

Figure 7.5: Borda evaluation problem encoding in *clingo*

*tribute/1*s representing three attributes, *numIss/1* the number of attributes, *threshold/1* the threshold value, together with *val/1*s the two values in the attributes' binary domains. Line 5 generates the search space of all alternatives over three binary attributes. It expresses that if X is an attribute, exactly one of *eval(X,Y)* holds for all *val(Y)*, i.e., exactly one value Y is assigned to X.

Let o be an alternative represented by a set of ground atoms $eval(i, x_i)$, one atom for each attribute X_i. Based on the representation of trees described above, for every tree v we get the set of ground atoms $vote(v, d_i^v, i, a_i)$. The Borda score of an alternative in that tree corresponds to the rank of the leaf the alternative leads to (in a "non-collapsed" tree), which is determined by the direction of descent (left or right) at each level. Roughly speaking, these directions give the binary representation of that rank, that is, the Borda score of the alternative. Let us define $s_B(v, o)$ as a function that computes the Borda score of alternative o given one vote v. Then one can check that

$$s_B(v, o) = \sum_{i=1}^{p} 2^{p-d_i^v} \cdot f(a_i, x_i), \tag{7.2}$$

where $f(a_i, x_i)$ returns 1 if $a_i = x_i$, 0 otherwise. Thus, to compute the Borda score with regard to a profile $\mathcal{V}$, we have

$$s_B(\mathcal{V}, o) = \sum_{v=1}^{n} \sum_{i=1}^{p} 2^{p-d_i^v} \cdot f(a_i, x_i). \tag{7.3}$$

In the program in Figure 7.5, lines 6 and 7 introduce predicate *wform/3* which computes $2^{p-d_i^v} \cdot f(a_i, x_i)$ used to compute Borda score. According to equation (7.3), if attribute I appears in vote V at depth D and A is its preferred value, and if the value of I is indeed A in an alternative o, then the weight W on I in V is 2^{P-D}, where P is the number of attributes; if attribute I is assigned the less preferred value in o, then the weight W on I in V is 0. The Borda score of the alternative is then equal to the sum of all the weights on every attribute in every vote, and this is computed using the aggregate function *#sum* built in the input language of *clingo* (rule 8). Rule 9 is an *integrity constraint* stating that contradiction is reached if predicate *goal/0* does

```
1  $domain(1..4).
2  attribute(1). attribute(2). attribute(3).
3  numIss(3).
4  val(0). val(1).
5  threshold(5).
6  1{ eval(I,M) : val(M) }1 :- attribute(I).
7  wform(V,I,W) :- vote(V,D,I,A), eval(I,A), numIss(P), W=#pow(2,P-D).
8  wform(V,I,0) :- vote(V,D,I,A), eval(X,M), A != M.
9  weight(V,I) $== W :- wform(V,I,W).
10 $sum{ weight(V,I) : voteID(V) : var(I) } $>= TH :- threshold(TH).
```

Figure 7.6: Borda evaluation problem encoding using *clingcon*

not hold in the solution. Together with rule 8, it is ensured that the Borda evaluation problem is satisfiable if and only if there is an answer set in which *goal/0* holds.

The encoding for the Borda winner problem for *clingo* replaces rules 7 and 8 in Figure 7.5 with the following single rule:

```
#maximize[ wform(V,I,W) = W ].
```

The *#maximize* statement is an optimization statement that maximizes the sum of all weights (W's) for which *wform(V,I,W)* holds.

Encoding the Borda evaluation problem in *clingcon*

In this encoding, we exploit *clingcon*'s ability to handle some numeric constraints by specialized constraint solving techniques (by means of the CP solver *Gecode* [71]). In Figure 7.6 we encode the Borda evaluation problem in *clingcon*.

Lines 2-8 are same as lines 1-7 in Figure 7.5. Line 9 defines the constraint variable *weight(V,I)* that assigns weight W to each pair (V,I) and line 10 defines a global

constraint by use of *$sum* declares that the Borda score must be at least the threshold. Line 1 restricts the domain of all constraint variables (only *weight/2* in this case) to [1,4] as weights of attributes in an LP-tree of 3 attributes are 2^0, 2^1 and 2^2.

The encoding for the Borda winner problem for *clingcon* replaces rules 10 in Figure 7.6 with the following one rule:

```
$maximize{weight(V,I):voteID(V):attribute(I)}.
```

The *$maximize* statement is an optimization statement that maximizes the sum over the set of constraint variables *weight(V,I)*.

Encoding the k-approval evaluation problem in *clingo*

One method to aggregate LP-trees according to k-approval can be designed reusing the Borda encodings for both problems and solvers. Given an alternative o, we can first compute $s_B(v, o)$ in every vote v and then compare $s_B(v, o)$ with $m - k$. If $s_B(v, o) \leq m - k$, $s_k(v, o) = 1$; otherwise, $s_k(v, o) = 0$. This method, however, is later turned out not quite effective for *clingo* in the sense that the rules to calculate Borda scores using aggregating predicate *#sum* result in large ground propositional theories that is hard for *clingo* to solve. We managed to work around this ineffectiveness by coming up with encodings using a heuristic that reduce the size of the ground programs for *clingo*. The heuristic is described in Theorem 24.

Theorem 24. *Given an LP-tree v and a positive integer k, we can construct in $O(p^2)$ time a Boolean formula ϕ of length $O(p^2)$ such that $s_k(v, o) = 1$ for an alternative o iff o satisfies ϕ.*

Proof. The algorithm is as follows.

1. ϕ is a disjunction of conjunctions of literals over attributes built as follows.

2. Compute the k-th preferred alternative $\vec{d}_k$ in time linear in p. Denote by $IO_{\vec{d}_k}$ the importance order $\vec{d}_k$ induces. Assume $IO_{\vec{d}_k} = X_{i_1} \rhd X_{i_2} \rhd \ldots \rhd X_{i_p}$.

3. The first conjunction $C_1 = l_{i_1} \wedge \ldots \wedge l_{i_j}$, where each l_{i_k}, $1 \leq k \leq j$, is X_{i_k} (resp. $\neg X_{i_k}$) if $\vec{d}_k(X_{i_j}) = 1_{i_j}$ (resp. $\vec{d}_k(X_{i_j}) = 0_{i_j}$).

4. For every attribute $X_{i_j} \in IO_{\vec{d}_k}$ such that $\vec{d}_k$ assigns it with its less preferred value (e.g., if $1_{i_j} > 0_{i_j}$, $\vec{d}_k(X_{i_j}) = 0_{i_j}$), we have a conjunction $C_{i_j} = l_{i_1} \wedge \ldots \wedge l_{i_{j-1}} \wedge l_{i_j}$, where each l_{i_k}, $1 \leq k \leq j-1$, is X_{i_k} (resp. $\neg X_{i_k}$) if $\vec{d}_k(X_{i_j}) = 1_{i_j}$ (resp. $\vec{d}_k(X_{i_j}) = 0_{i_j}$) and l_{i_j} is X_{i_k} (resp. $\neg X_{i_k}$) if $\vec{d}_k(X_{i_j}) = 0_{i_j}$ (resp. $\vec{d}_k(X_{i_j}) = 1_{i_j}$).

□

In order to compute the k-th preferred alternative $\vec{d}_k$, we need some auxiliary predicates to help the computation. We define predicates *voteK/4* and *evalK/4* that are basically copies of *vote/4* and *eval/4* in the logic representation of LP-trees except that *evalK/4* describes $\vec{d}_k$. A predicate *evalK(V,D,I,M)* means that in vote V the k-th ranked alternative assigns value M to attribute I at depth D. For the example LP-tree in Figure 3.3, we have the follow ancillary logic program in Figure 7.7.

```
1  voteK(1,1,1,1).
2  voteK(1,2,2,1) :- evalK(1,1,1,1).
3  voteK(1,3,3,1) :- evalK(1,2,2,1), evalK(1,1,1,1).
4  voteK(1,3,3,0) :- evalK(1,2,2,0), evalK(1,1,1,1).
5  voteK(1,2,3,0) :- evalK(1,1,1,0).
6  voteK(1,3,2,0) :- evalK(1,1,1,0).
```

Figure 7.7: Auxiliary data in logic rules for computing $\vec{d}_k$

We now present the encoding of the k-Apprvoal evaluation problem in *clingo* (Figure 7.8), where $k = 5$.

```
1  attribute(1). attribute(2). attribute(3).
2  numIss(3).
3  val(0). val(1).
4  k(1,1). k(2,0). k(3,0).
5  threshold(5).
6  evalK(V,D,I,M) :- vK(V,D,I,M), k(D,0).
7  evalK(V,D,I,1-M) :- vK(V,D,I,M), k(D,1).
8  1{ eval(I,M) : val(M) }1 :- attribute(I).
9  rank(V,1) :- vote(V), numIss(N),
                N{eval(I,M) : evalK(VV,D,I,M) : V==VV}N.
10 rank(V,1) :- vote(V), k(D,1),
                D-1{eval(I,M) : evalK(VV,DD,I,M) : A==AA : DD<=D-1}D-1,
                1{eval(I,M) : evalK(V,D,I,MM) : M!=MM}1.
11 goal :- S = #sum [ rank(V,Y) = Y ], threshold(TH), S >= TH.
12 :- not goal.
```

Figure 7.8: k-Approval evaluation problem encoding in *clingo*

7.5 The Problems in Weighted Partial Maximum Satisfiability

In this section, we call "the evaluation and the winner problems based on a positional scoring rule r" by "the r problems." We show an algorithm that translate the positional scoring rule problems into Weighted Partial Maximum Satisfiability instances. We first translate the positional scoring rule problems to Weighted Terms Maximum Satisfiability instances, which are then transformed into Weighted Partial Maximum Satisfiability instances.

Weighted Partial Maximum Satisfiability

Definition 37. Let X be a set of boolean variables $\{X_1, \ldots, X_q\}$, Ψ a set of weighted terms of the form

$$\{(t_1, w_1), \ldots, (t_n, w_n)\},$$

where each term is a conjunction of literals on X. The *Weighted Terms Maximum Satisfiability* (WTM) problem is to find an assignment of X that maximizes the sum of the weights of the satisfied terms in Ψ.

Definition 38. Let X be a set of boolean variables $\{X_1, \ldots, X_p\}$, a *weighted partial formula* Φ [4] is a multiset of weighted clauses over X of the form

$$\{(c_1, w_1), \ldots, (c_n, w_n), (c_{n+1}, w_{n+1}), \ldots, (c_{n+m}, w_{n+m})\},$$

where each w_i, $1 \leq i \leq n$, is a positive integer and $w_{n+1} = \ldots = w_{n+m} = \sigma = 1 + \sum_{i=1}^{n} w_i$. Clause c_j is *hard* if $w_j = \sigma$; *soft*, otherwise.

Definition 39. Let X be a set of boolean variables $\{X_1, \ldots, X_p\}$, Φ a weighted partial formula, the *Weighted Partial Maximum Satisfiability* (WPM) problem is to find an assignment of X that maximizes the sum SW of weights of satisfied clauses in Φ. If $SW < m * \sigma$, it means that at least one hard clause is fasified and we say that Φ is *unsatisfiable*.

Clearly, the WPM problem generalizes the SAT problem [39], the MAXSAT problem [26] and the Partial MAXSAT problem [26].

Translating a WTM instance into an equivalent WPM instance

Now we show how a WTM instance Ψ can be translated into a WPM instance Φ in polynomial time such that a solution to Φ projected onto Ψ's alphabet X_Ψ is a solution to Ψ.

Theorem 25. *Given a WTM instance Ψ, a WPM instance Φ can be computed in time $O(nq)$ such that any solution to Φ restricted to X_Ψ is a solution to Ψ.*

Proof. The tranlation algorithm is detailed in Algorithm 4.

[4] This definition is slightly adapted of the commonly used [6, 5].

Let X_Ψ be $\{X_1, \ldots, X_q\}$, X_Φ be $\{X_1, \ldots, X_q, C_1, \ldots, C_n\}$. Assume v is a solution to the WPM instance Φ over X_Φ, we show that the restriction, $v|_{X_\Psi}$, is a solution to the original WTM instance Ψ. Let $S = \{t_{i_1}, \ldots, t_{i_s}\}$ be the set of terms in Ψ satisfied by v (or, equivalently, $v|_{X_\Psi}$). It is clear that v satisfies $\{C_{i_k} : t_{i_k} \in S\}$ and falsifies $\{C_{i_k} : t_{i_k} \notin S\}$; since, otherwise, v would not have the maximal sum SW for Φ. Denote by x_i the number of literals in term t_i. Let v' be an arbitrary assignment such that $SW_{v'} < SW_v$, and $S' = \{t_{j_1}, \ldots, t_{j_r}\}$ the set of terms in Ψ satisfied by v'.

According to Algorithm 4, we have $SW_v = \sum_{k=1}^{s} w_{i_k} + \sum_{k=1}^{n} \sigma + \sum_{k=1}^{n}(\sum_{o=1}^{x_k} \sigma)$ and $SW_{v'} = \sum_{k=1}^{r} w_{j_k} + \sum_{k=1}^{n} \sigma + \sum_{k=1}^{n}(\sum_{o=1}^{x_k} \sigma)$. Then, we have $SW_v - SW_{v'} = \sum_{k=1}^{s} w_{i_k} - \sum_{k=1}^{r} w_{j_k} > 0$. Thus, we know $v|_{X_\Psi}$ is a solution to the original WTM instance Ψ. □

Algorithm 4: Compute equivalent WPM instances from WTM instances

Input: a WTM instance Ψ
Output: an equivalent WPM instance Φ

$\Phi \leftarrow \emptyset$;
$\sigma \leftarrow 1 + \sum_{i=1}^{n} w_i$;
foreach $(t_i, w_i) \in \Psi$ **do**
    introduce a new variable C_i and $\Phi \leftarrow \Phi \cup (C_i, w_i)$;
    $\Phi \leftarrow \Phi \cup (C_i \vee \bigvee_{l_j \in t_i} \neg l_j, \sigma)$;
    foreach $l_j \in t_i$ **do**
        $\Phi \leftarrow \Phi \cup (\neg C_i \vee l_j, \sigma)$;
    end
end
return Φ

Encoding Borda problems in WTM and WPM

The LP-tree in Figure 3.3 under Borda is tranlated to a WTM instance in Figure 7.9.

Then the WTM instance in Figure 7.9 is transformed into a WPM instance in Figure 7.10.

$(X_1, 4)$
$(X_1 \wedge X_2, 2)$
$(X_1 \wedge X_2 \wedge X_3, 1)$
$(X_1 \wedge \neg X_2 \wedge \neg X_3, 1)$
$(\neg X_1 \wedge \neg X_3, 2)$
$(\neg X_1 \wedge \neg X_2, 1)$

Figure 7.9: The WTM instance of the LP-tree v

$(C_1, 4)$
$(\neg C_1 \vee X_1, 12)$
$(\neg X_1 \vee C_1, 12)$
$(C_2, 2)$
$(\neg C_2 \vee X_1, 12)$
$(\neg C_2 \vee X_2, 12)$
$(\neg X_1 \vee \neg X_2 \vee C_2, 12)$
$(C_3, 1)$
$(\neg C_3 \vee X_1, 12)$
$(\neg C_3 \vee X_2, 12)$
$(\neg C_3 \vee X_3, 12)$
$(\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee C_3, 12)$
$(C_4, 1)$
$(\neg C_4 \vee X_1, 12)$
$(\neg C_4 \vee \neg X_2, 12)$
$(\neg C_4 \vee \neg X_3, 12)$
$(\neg X_1 \vee X_2 \vee X_3 \vee C_4, 12)$
$(C_5, 2)$
$(\neg C_5 \vee \neg X_1, 12)$
$(\neg C_5 \vee \neg X_3, 12)$
$(X_1 \vee X_3 \vee C_5, 12)$
$(C_6, 1)$
$(\neg C_6 \vee \neg X_1, 12)$
$(\neg C_6 \vee \neg X_2, 12)$
$(X_1 \vee X_2 \vee C_6, 12)$

Figure 7.10: The WPM instance of the LP-tree v

Encoding k-approval problems in WTM and WPM

The LP-tree in Figure 3.3 under 5-Approval is translated to a WTM instance in Figure 7.11.

$(\neg X_1 \wedge \neg X_2 \wedge \neg X_3, 1)$

$(X_1, 1)$

Figure 7.11: The WTM instance of the LP-tree v under 5-Approval

Then the WTM instance in Figure 7.11 is transformed into a WPM instance in Figure 7.12.

$(C_1, 1)$

$(\neg C_1 \vee \neg X_1, 3)$

$(\neg C_1 \vee \neg X_2, 3)$

$(\neg C_1 \vee \neg X_3, 3)$

$(X_1 \vee X_2 \vee X_3 \vee C_1, 3)$

$(C_2, 1)$

$(\neg C_2 \vee X_1, 3)$

$(\neg X_1 \vee C_2, 3)$

Figure 7.12: The WPM instance of the LP-tree v under 5-Approval

7.6 Experiments

Here we present and analyze the experimental results from solving the Winner problem and the Evaluation problem using two Answer Set Programming solvers *clingo* (version 4.2.1) and *clingcon* (version 2.0.3) and one Constraint Satisfaction Problem solver *toulbar* (version 0.9.6.0-dev).

All our experiments were performed on a machine with an Intel(R) Core(TM) i7 CPU @ 2.67GHz and 8 GB RAM running Ubuntu 12.04 LTS.

We first consider the winner problem. In the study, we consider the computation time with a fixed number of attributes (5/10/20) and for each number of attributes

we range the number of votes in a profile up to 3000 for $\{Borda, 2^{p-2}\text{-}approval, 2K\text{-}approval\} \times \{clingcon, clingo, toulbar\}$. Then we fix the number of votes (1000) and vary the number of attributes up to 20, again for same set of settings. Each time result in seconds is computed as the mean of 20 tests over different randomly generated profiles of LP-trees.

Structure of the Simple LP Trees

To experiment with the programs presented above and with *clingo* and *clingcon* solvers, we generate logic programs that represent random LP-trees and profiles of random LP-trees. Our algorithm generates encodings of trees from the most general class CI-CP under the following restrictions: (1) Each LP-tree has exactly two paths with the splitting node appearing at depth $d_s = \lfloor \frac{p}{2} \rfloor$; (2) Each non-root node at depth $\leq d_s + 1$ has exactly one parent; (3) Each node at depth $> d_s + 1$ has exactly two parents, one of which is at depth $< d_s$.[5]

The algorithm starts by randomly selecting attributes to label the nodes on the path from the root to the splitting node and then, similarly, labels the nodes on each of the two paths (different labelings can be produced for each of them). Then, for each non-root node, the algorithm selects at random one or two parent nodes (as appropriate based on the location of the node). Finally, the algorithm decides local preferences (for each combination of values of the parent attributes) randomly picking one over the other. In each step, all possible choices are equally likely. We call CI-CP LP-trees satisfying these restrictions *simple*. Each simple LP-tree has size linear in p. Figure 7.13 depicts a CI-CP tree of 4 attributes in this class.

[5]The restrictions are motivated by the size of the representation considerations. They ensure that the size of generated LP-trees is linear in the number of attributes.

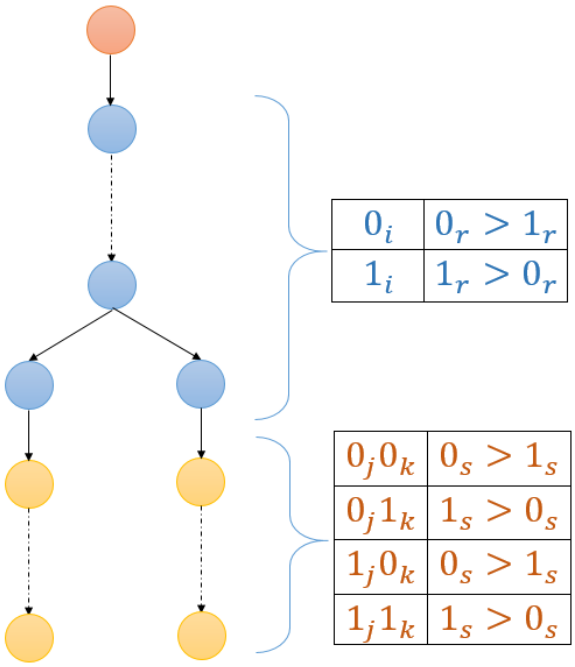


Figure 7.13: *Simple* CI-CP tree

Solving the Winner Problem

We refer to 7.14 for the empirical results on solving the Winner problem. Each point in a figure represents an average of computation time spent on solving 20 different Winner problem instances given randomly generated LP profiles.

It is clear that our experiments on the winner problem for the three voting rules with fixed number of attributes are consistent with the property that the problem is solvable in polynomial time. All three solvers scale up well. Figures 7.14(a),(c) and (e) depict the result for the cases with 10 attributes. When we fix the number of votes and vary the number of attributes the time grows exponentially with p (cf. Figures 7.14(b),(d) and (f)), again consistently with the computational complexity of the problems (NP-hardness).

Generally *clingo* is better compared to *clingcon* in solving the winner problem for the three scoring rules. Moreover, *clingo* outperforms *toulbar* in solving the winner problem for the Borda rule, while *toulbar* performs better than *clingo* for the two

approval rules.

For the winner problems, our experiments demonstrates that profiles of LP-trees of practical sizes can be effectively handled by our solvers, up to 3000 votes per profile over up to 20 attributes. But going beyond 20 attributes remains a challenge.

Solving the Evaluation Problem

The evaluation problem can be reduced to the winner problem, as an evaluation problem instance has an answer *YES* if and only if the score of the winner equals or exceeds the threshold. Thus, the evaluation problem is at most as complex as the winner problem.

For the evaluation problem, we compare its experimental complexity with that of the winner problem. For each of the 20 randomly generated profiles of 1000 votes, we compute the winning score WS and set the threshold for the evaluation problem with a percentage of WS, starting with 5% and incremented by 5% for the following tests until we reach the full value of WS. We run one more test with the threshold $WS + 1$ (there is no solution then and the overall method allows for the experimental comparison of the hardness of the winner and evaluation problems). That allows us to study the effectiveness of solvers. We again present and compare average time results.

First, we note that for *clingo*, the evaluation problem is harder than the winner problem in the entire range for Borda (Figure 7.15(c)). We attribute that to the fact that the encodings of the evaluation problem have to model the threshold constraint with the *#sum* rule which, in *clingo*, leads to large ground theories that it finds hard to handle. In the winner problem encodings, the *#sum* rule is replaced with an optimization construct, which allows us to keep the size of the ground theory low.

Second, we notice that, except for Borda and *clingo*, the evaluation problem is easier than the winner problem when the threshold values are smaller than the win-

ning score and the evaluation problem becomes harder when the thresholds are close to it. We refer to Figures 7.15(a), (b) and (c).

Thirdly, in all cases *clingcon* outperforms *clingo* on the evaluation problems (cf. Figures 7.15(a), (b) and (c)). It is especially clear for Borda, where the range of scores is much larger than in the case of approval rules. That poses a challenge for *clingo* that instantiates the *#sum* rule over that large range, which *clingcon* is able to avoid.

Finally, we compare the effectiveness of *clingcon* and *toulbar* on solving the evaluation problems. Generally, *clingcon* performs better for the Borda rule, whereas *toulbar* is better for the two approval rules. Again, to see this, we refer to Figures 7.15(a), (b) and (c).

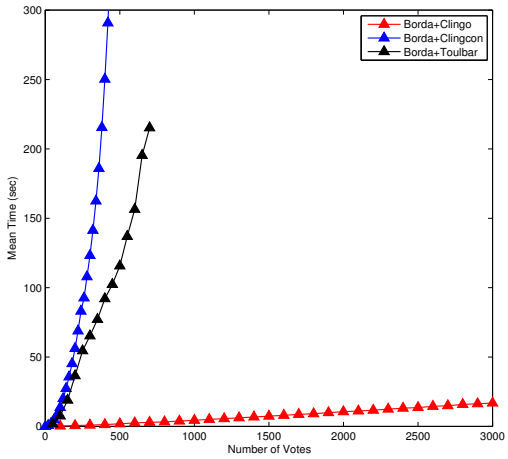
7.7 Conclusions

Aggregating votes expressed as LP-trees is a rich source of interesting theoretical and practical problems. In particular, the complexity of the winner and evaluation problems for scoring rules is far from being fully understood. First results on the topic were provided by Lang et al. [53]; our work exhibited another class of positional scoring rules for which the problems are NP-hard and NP-complete, respectively. However, a full understanding of what makes a positional scoring rule hard remains an open problem.

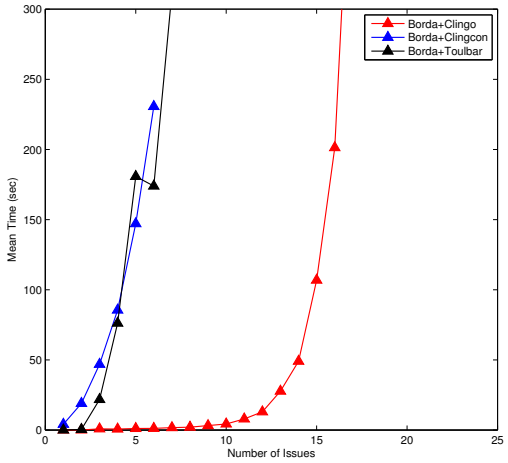
Importantly, our results show that ASP tools are effective in modeling and solving the winners and the evaluation problems for some positional scoring rules such as Borda, 2^{p-2}-approval and $2K$-approval. When the number of attributes is fixed the ASP tools scale up consistently with the polynomial time complexity. In general, the tools are practical even if the number of attributes is up to 15 and the number of votes is as high as 500. This is remarkable as 15 binary attributes determine the space of over 30,000 alternatives.

Finally, the preference aggregation problems form interesting benchmarks for ASP tools that stimulate advances in ASP solver development. As the preference aggregation problems involve large domains, they put to the test those features of ASP tools that attempt to get around the problem of grounding programs over large domains. Our results show that the optimization statements in *clingo* in general perform well. When they cannot be used, as in the evaluation problem, it is no longer the case. The solver *clingcon*, which reduces grounding and preprocessing work by delegating some tasks to a constraint solver, performs well in comparison to *clingo* on the evaluation problem, especially for the Borda rule (and we conjecture, for all rules that result in large score ranges).

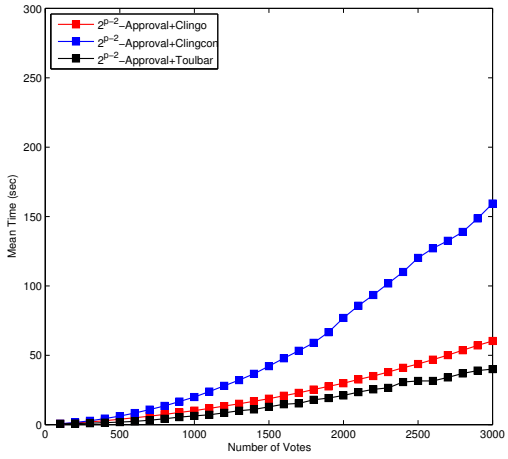
In the future work we will expand our experimentation by developing methods to generate richer classes of randomly generated LP-trees. We will also consider the use of ASP tools to aggregate votes given in other preference systems such as CP-nets [17] and answer set optimization (ASO) preferences [24].



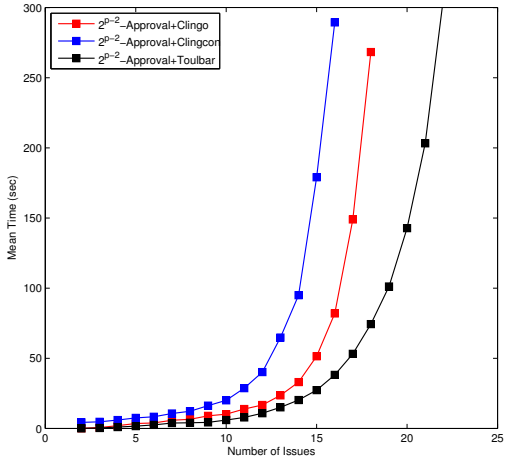
(a) Borda, fixed #attributes(10)



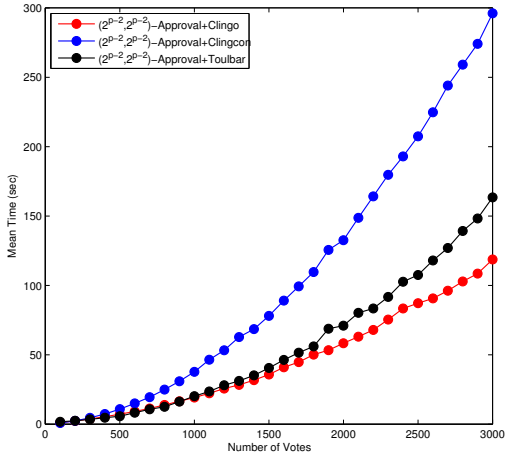
(b) Borda, fixed #votes(1000)



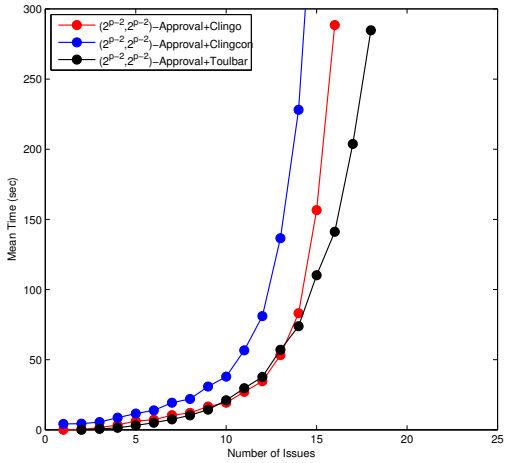
(c) 2^{p-2}-Approval, fixed #attributes(10)



(d) 2^{p-2}-Approval, fixed #votes(1000)

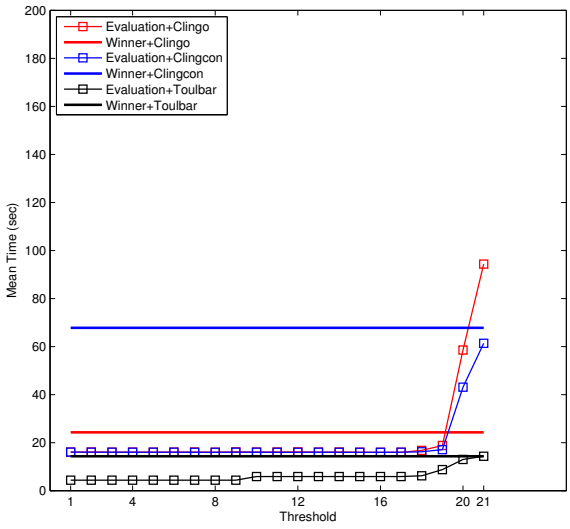


(e) $2K$-Approval, fixed #attributes(10)

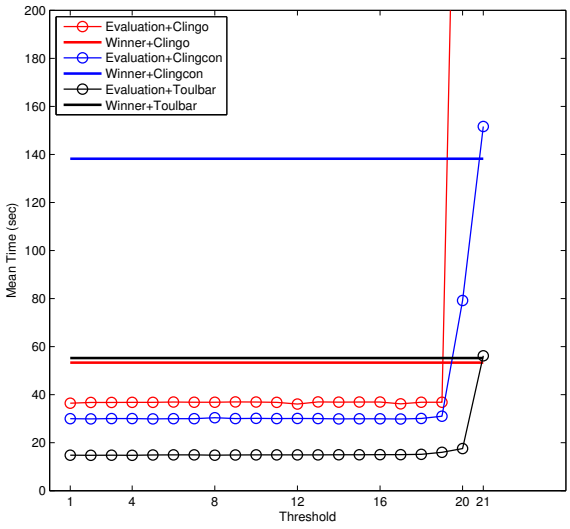


(f) $2K$-Approval, fixed #votes(1000)

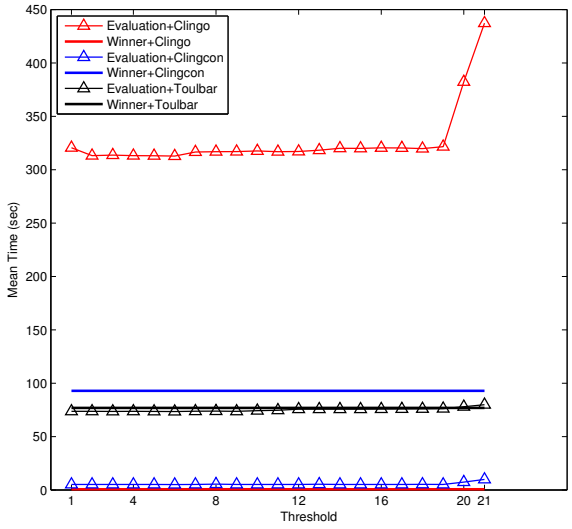
Figure 7.14: Solving the winner problem given *simple* LP-trees



(a) 2^{p-2}-Approval (1000votes/13attributes)



(b) $2K$-Approval (1000votes/13attributes)



(c) Borda (1000votes/4attributes)

Figure 7.15: Solving the evaluation problem given *simple* LP-trees

Chapter 8 Summary

I will now discuss my ongoing research work, as well as point out directions of future research.

My ongoing research work that, once done, would be included in the final version of my dissertation contains the following components:

1. Preference learning and approximation:
    - Learning forests of lexicographic preference trees.
    - Approximating CP-nets using lexicographic preference trees.
2. Preference reasoning:
    - Aggregating partial lexicographic preference trees (PLP-trees).
    - Preference misrepresenting in elections where votes are LP-trees or PLP-trees.

Regarding future research directions, I will establish a research program of data-driven preference engineering, including data-driven preference learning, and preference reasoning and applications.

For data-driven preference learning, I plan to study methods and algorithms below.

1. Recommender Systems[2]:
    - Collaborative
    - Content-based
    - Hybrid
2. Machine Learning (fitting function):

- Supervised learning (e.g., decision trees, random forests)
- Label ranking[48]

3. Model-based Learning (learning interpretable decision models):

- Preference Elicitation (Human-in-the-Loop)
- Conditional Preference Networks, Preference Trees
- Stochastic Models (e.g., Choquet integral[74], TOPSIS-like models[3])

For preference reasoning and applications, I propose to investigate both theoretic and practical problems in areas as follows.

1. Social Choice and Welfare[8, 20]:

- Voting
- Fair division
- Strategyproof Social Choice

2. Automated Planning and Scheduling[72, 15, 14]:

- Travel scheduling
- Manufacturing
- Traffic control

Bibliography

[1] Kenneth J. Aarrow. *Social Choice and Individual Values*. John Wiley and Sons, 1951.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 2005.

[3] Manish Agarwal, Ali Fallah Tehrani, and Eyke Hüllermeier. Preference-based learning of ideal solutions in topsis-like decision models. *Journal of Multi-Criteria Decision Analysis*, 2014.

[4] D Allouche, S de Givry, and T Schiex. Toulbar2, an open source exact cost function network solver. Technical report, Technical report, INRIA, 2010.

[5] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. Solving (weighted) partial maxsat through satisfiability testing. In *Theory and Applications of Satisfiability Testing*, pages 427–440. 2009.

[6] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial maxsat. In *AAAI*, 2010.

[7] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[8] Kenneth J Arrow, Amartya Sen, and Kotaro Suzumura. *Handbook of Social Choice and Welfare*, volume 1 & 2. 2010.

[9] K.J. Arrow, A. Sen, and K. Suzumura. *Handbook of Social Choice and Welfare, Vol 1*. Elsevier, 2002.

[10] Fahiem Bacchus and Adam J. Grove. Graphical models for preference and utility. In Philippe Besnard and Steve Hanks, editors, *UAI*, pages 3–10. Morgan Kaufmann, 1995.

[11] J.J. Bartholdi, C.A. Tovey, and M. A. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6:227–241, 1989.

[12] J.J. Bartholdi, C.A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.

[13] J.J. Bartholdi, C.A. Tovey, and M. A. Trick. How hard is it to control an election? *Mathl. Comput. Modelling (Special Issue on Formal Theories of Politics*, 16(8/9):27–40, 1992.

[14] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. *arXiv preprint*, 2015.

[15] Meghyn Bienvenu, Christian Fritz, and Sheila A McIlraith. Specifying and computing preferred plans. *Artificial Intelligence*, 2011.

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

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

[18] Ronen I. Brafman and Carmel Domshlak. Introducing variable importance tradeoffs into cp-nets. In *UAI*, pages 69–76, 2002.

[19] Steven J. Brams and Peter C. Fishburn. *Approval Voting*. Springer-Verlag, 2007.

[20] Felix Brandt, Vincent Conitzer, , and Ulle Endriss. *Computational social choice.* MIT Press., 2012.

[21] Michael Bräuning and H Eyke. Learning conditional lexicographic preference trees. *Preference learning: problems and applications in AI*, 2012.

[22] Gerhard Brewka. Complex preferences for answer set optimization. In *KR*, pages 213–223, 2004.

[23] Gerhard Brewka, Salem Benferhat, and Daniel Le Berre. Qualitative choice logic. *Artificial Intelligence*, 157(1):203–237, 2004.

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

[25] Gerhard Brewka, Ilkka Niemela, and Miroslaw Truszczynski. Answer set optimization. In *PROC. IJCAI-03*, pages 867–872. Morgan Kaufmann, 2003.

[26] David Cohen, Martin Cooper, and Peter Jeavons. A complete characterization of complexity for boolean constraint optimization problems. In *Principles and Practice of Constraint Programming*, pages 212–226. 2004.

[27] Florence Dupin De Saint-Cyr, Jérôme Lang, and Thomas Schiex. Penalty logic and its link with dempster-shafer theory. In *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*, 1994.

[28] József Dombi, Csanád Imreh, and Nándor Vincze. Learning lexicographic orders. *European Journal of Operational Research*, 183:748–756, 2007.

[29] Carmel Domshlak, Eyke Hllermeier, Souhila Kaci, and Henri Prade. Preferences in ai: An overview. *Artificial Intelligence*, 175(7-8):1037 – 1052, 2011.

[30] Didier Dubois, Jérôme Lang, and Henri Prade. A brief overview of possibilistic logic. In *ECSQARU*, pages 53–57, 1991.

[31] J. Duggan and T. Schwartz. Strategic manipulability without resoluteness or shared beliefs: Gibbard-satterthwaite generalized. *Social Choice and Welfare*, 17:85–93, 2000.

[32] Hélène Fargier, Vincent Conitzer, Jérôme Lang, Jérôme Mengin, and Nicolas Schmidt. Issue-by-issue voting: an experimental evaluation. In *MPREF*, 2012.

[33] Peter C. Fishburn. *The Theory of Social Choice*. Princeton University Press, 1973.

[34] Peter C. Fishburn. Axioms for lexicographic preferences. *The Review of Economic Studies*, 42:415–419, 1975.

[35] Niall M Fraser. Applications of preference trees. In *Proceedings of IEEE Systems Man and Cybernetics Conference*, pages 132–136. IEEE, 1993.

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

[37] Johannes Fürnkranz and Eyke Hüllermeier. Preference learning: An introduction. In *Preference Learning*, pages 1–17. Springer, 2011.

[38] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

[39] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[40] William I Gasarch. The P =? NP poll. *Sigact News*, 33(2):34–47, 2002.

[41] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):105–124, 2011.

[42] Thomas Eiter Gerhard Brewka and Miroslaw Truszczynski. Answer set programming at a glance.

[43] A. Gibbard. Manipulation of voting schemes: A general result. *Econometrica*, 41:587–601, 1973.

[44] Judy Goldsmith, Jrme Lang, Miroslaw Truszczyski, and Nic Wilson. The computational complexity of dominance and consistency in cp-nets. In *In Proceedings of IJCAI-05*, pages 144–149, 2005.

[45] Christophe Gonzales and Patrice Perny. GAI Networks for Utility Elicitation. In *Proceedings of the 9th International Conference on the Principles of Knowledge Representation and Reasoning*, pages 224–234, 2004. INT LIP6 DECISION.

[46] Peter Haddawy and Steve Hanks. Representations for decision-theoretic planning: Utility functions for deadline goals. *KR*, 92:71–82, 1992.

[47] James L Hein. *Discrete Structures, Logic, and Computability*. Jones and Bartlett Publishers, LLC, 2010.

[48] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 2008.

[49] Souhila Kaci. *Working with Preferences: Less Is More*. Cognitive Technologies. Springer, 2011.

[50] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani. The minimum satisfiability problem. *SIAM J. Discrete Math.*, 7(2):275–283, 1994.

[51] Mark W. Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.*, 36(3):490–509, 1988.

[52] Jérôme Lang, Jérôme Mengin, and Lirong Xia. Aggregating conditionally lexicographic preferences on multi-issue domains. In *CP*, pages 973–987, 2012.

[53] Jérôme Lang, Jérôme Mengin, and Lirong Xia. Aggregating conditionally lexicographic preferences on multi-issue domains. In *CP*, pages 973–987, 2012.

[54] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1997.

[55] Xudong Liu. Aggregating lexicographic preference trees using answer set programming: Extended abstract. In *The 23rd International Joint Conference on Artificial Intelligence Doctoral Consortium, (IJCAI-DC)*, 2013.

[56] Xudong Liu and Miroslaw Truszczynski. Aggregating conditionally lexicographic preferences using answer set programming solvers. In *Algorithmic Decision Theory*, volume 8176, pages 244–258. Springer, 2013.

[57] Xudong Liu and Miroslaw Truszczynski. Aggregating conditionally lexicographic preferences using answer set programming solvers. In *ADT*, pages 244–258, 2013.

[58] Xudong Liu and Miroslaw Truszczynski. Preference trees: A language for representing and reasoning about qualitative preferences. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

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

[60] Xudong Liu and Miroslaw Truszczynski. Reasoning with preference trees over combinatorial domains. In *Proceedings of the 4th International Conference on Algorithmic Decision Theory (ADT)*, volume 9346, pages 19–34. Springer, 2015.

[61] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.

[62] V.W. Marek and M. Truszczynski. Stable models and an alternative logic programming paradigm. In K.R. Apt, V.W. Marek, M. Truszczynski, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer, Berlin, 1999.

[63] K.O. May. A Set of Independent Necessary and Sufficient Conditions for Simple Majority Decision. *Econometrica*, 20(4):680–684, 1952.

[64] H. Moulin. *Axioms of Cooperative Decision Making*. Cambridge University Press, 1988.

[65] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[66] Max Ostrowski and Torsten Schaub. ASP modulo CSP: The clingcon system. *TPLP*, 12(4-5):485–503, 2012.

[67] C. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Inc., 1994.

[68] Christos Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, pages 229–234, New York, NY, USA, 1988. ACM.

[69] M. Satterthwaite. Strategy-proofness and Arrow's Conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10:187–216, 1975.

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

[71] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with gecode, 2010.

[72] Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 2006.

[73] Yves Sprumont. The division problem with single-peaked preferences: a characterization of the uniform allocation rule. *Econometrica*, 59(2):509–519, 1991.

[74] Ali Fallah Tehrani, Weiwei Cheng, and Eyke Hüllermeier. Choquistic regression: Generalizing logistic regression using the choquet integral. In *EUSFLAT*, 2011.

[75] Wikipedia. Social choice theory, 2013.

[76] Nic Wilson. Extending cp-nets with stronger conditional preference statements. In *in Proceedings of AAAI-04*, pages 735–741, 2004.

[77] Lirong Xia and Vincent Conitzer. Approximating common voting rules by sequential voting in multi-issue domains. In *ISAIM*, 2012.

[78] 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.

[79] Ying Zhu and Miroslaw Truszczynski. On optimal solutions of answer set optimization problems. In *Logic Programming and Nonmonotonic Reasoning*, pages 556–568, 2013.

Xudong Liu

http://www.cs.uky.edu/∼liu/ • liu@cs.uky.edu

Research Interests

My research focuses on exciting topics in artificial intelligence and social science, including preferences (i.e., preference modeling, learning and reasoning), data-driven and computational social science, knowledge representation and reasoning, decision theory, data mining, and machine learning.

Education

University of Kentucky USA
Doctor of Philosophy, computer science *Aug. 2010 – May 2016*
GPA:**4.00**/4.00
Advisor: Dr. Miroslaw Truszczynski

- **Courses**: Modern Operating Systems, Numerical Analysis, Algorithm Design, Distributed Operating System, Satisfiability and Equivalence Checking, Database Systems, Computer Networks, Networks Security, Preferences in AI and Decision Theory, Principles of Constraint Satisfaction, Comparative Decision Making, Preparing Future Faculty, and Grant Writing.

Harbin Institute of Technology China
Bachelor of Engineering, software engineering *Aug. 2006 – July 2010*
GPA:3.56/4.00
Advisor: Prof. Yushan Sun

Employment

R&D Intern Palo Alto Research Center (PARC), USA
Supervisor: Dr. Christian Fritz *Jun. 2015 – Aug. 2015*

- Conducted research on and developed system modules for representing and reasoning about user constraints and preferences in trip planning.

Graduate Research Assistant University of Kentucky, USA
Advisor: Dr. Miroslaw Truszczynski *Aug. 2010 – May 2015*

- Conducted research on logic-based knowledge representation formalisms, studied and built tools for representing and reasoning about constraints and preferences in artificial intelligence.

Graduate Teaching Assistant University of Kentucky, USA
Advisors: Dr. Truszczynski, Dr. Pike and Dr. Moore *Aug. 2010 – May 2015*

- CS215 - Introduction to Program Design and Problem Solving: leading lab sessions, grading assignments, preparing exam questions and solutions, and holding office hours.

- CS375 - Logic and Theory of Computing: grading assignments, preparing assignment solutions, and holding office hours.
- CS463G - Introduction to Artificial Intelligence: guest instructor, on knowledge representation and reasoning, e.g., propositional logic, and first-order logic.

Undergraduate Teaching Assistant Harbin Institute of Technology, China
Advisor: Prof. Yushan Sun *Aug. 2008 – May 2010*

- Compilers and J2EE, including leading lab sessions and grading assignments.

Undergraduate Intern Information Security Lab, Harbin Institute of Technology, China
Advisor: Prof. Yushan Sun *Aug. 2009 – May 2010*

- Used PHP, MySQL and Apache Tomcat to implement modules of a management system.

Professional Services

Student member: Association for the Advancement of Artificial Intelligence (AAAI)

Student volunteer: 29th AAAI Conference on Artificial Intelligence (AAAI-15)

Program committee: IJCAI-16, IJCAI-13

Paper reviewer: JAIR, AAAI-14, ISAIM-14

Local arrangement committee: ADT-15, LPNMR-15, ICLP-11, NonMon@30-10

Refereed Publications

Conferences:

1. **Xudong Liu**. *Modeling, Learning and Reasoning with Qualitative Preferences*. In Proceedings of the 4th International Conference on Algorithmic Decision Theory (ADT), volume 9346, pages 587-592, 2015. Springer
2. **Xudong Liu** and Miroslaw Truszczynski. *Reasoning with Preference Trees over Combinatorial Domains*. In Proceedings of the 4th International Conference on Algorithmic Decision Theory (ADT), volume 9346, pages 19-34, 2015. Springer
3. **Xudong Liu** and Miroslaw Truszczynski. *Learning Partial Lexicographic Preference Trees over Combinatorial Domains*. In Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI), pages 1539-1545, 2015. AAAI Press
4. **Xudong Liu** and Miroslaw Truszczynski. *Aggregating Conditionally Lexicographic Preferences Using Answer Set Programming Solvers*. In Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT), volume 8176, pages 244-258, 2013. Springer
5. Matthew Spradling, Judy Goldsmith, **Xudong Liu**, Chandrima Dadi and Zhiyu Li. *Roles and Teams Hedonic Game*. In Proceedings of the 3rd International Conference on Algorithmic Decision Theory (ADT), volume 8176, pages 351-362, 2013. Springer

Workshops:

1. **Xudong Liu** and Miroslaw Truszczynski. *Preference Trees: A Language for Representing and Reasoning about Qualitative Preferences*. In Proceedings of the 8th AAAI Multidisciplinary Workshop on Advances in Preference Handling (MPREF), pages 55-60, 2014. AAAI Press

Abstract:

1. **Xudong Liu**. *Aggregating Lexicographic Preference Trees Using Answer Set Programming: Extended Abstract*. In 23rd International Joint Conference on Artificial Intelligence Doctoral Consortium (IJCAI DC), 2013.

Technical Talks

1. *Personalization in Trip Planning*. Oral. At the Keeping Current Seminar, Department of Computer Science, University of Kentucky, 2015.
2. *Preference and Social Choice over Combinatorial Domains*. Oral. At the 1st ADT/LPNMR Doctoral Consortium (ADT/LPNMR-DC-15), Lexington, Kentucky, USA, 2015.
3. *Modeling, Learning and Reasoning with Qualitative Preferences*. Poster. At the 1st ADT/LPNMR Doctoral Consortium (ADT/LPNMR-DC-15), Lexington, Kentucky, USA, 2015.
4. *Reasoning with Preference Trees over Combinatorial Domains*. Oral. At the 4th International Conference on Algorithmic Decision Theory (ADT-15), Lexington, Kentucky, USA, 2015.
5. *On Personalizability and Extensibility of Multi-Modal Trip Planning*. Oral. At Dialog, Palo Alto Research Center, Palo Alto, California, USA, 2015.
6. *Constraints and Preferences in Multi-modal Trip Planning*. Poster. At the PARC Summer Intern Poster Session, Palo Alto Research Center, Palo Alto, California, USA, 2015.
7. *Learning Partial Lexicographic Preference Trees over Combinatorial Domains*. Oral (ratio: 12%=238/1991). At the 29th AAAI Conference on Artificial Intelligence (AAAI-15), Austin, Texas, USA, 2015.
8. *Preference Trees: A Language for Representing and Reasoning about Qualitative Preferences*. Oral. At the 8th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-14), Quebec City, Canada, 2014.
9. *Aggregating Conditionally Lexicographic Preferences Using Answer Set Programming Solvers*. Oral. At the 3rd International Conference on Algorithmic Decision Theory (ADT-13), Universite lebre de Bruxelles, Belgium, 2013.
10. *Reasoning About Lexicographic Preferences Over Combinatorial Domains*. At the Keeping Current Seminar, Department of Computer Science, University of Kentucky, 2013.
11. *Roles and Teams Hedonic Game*. Oral. At the 7th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-13), Tsinghua University, Beijing, China, 2013.

12. *Aggregating Conditionally Lexicographic Preferences Using Answer Set Programming Solvers*. Oral. At the 7th Multidisciplinary Workshop on Advances in Preference Handling (MPREF-13), Tsinghua University, Beijing, China, 2013.
13. *Aggregating Lexicographic Preference Trees Using Answer Set Programming: Extended Abstract*. Poster. At the 23rd International Joint Conference on Artificial Intelligence Doctoral Consortium (IJCAI-DC-13), Tsinghua University, Beijing, China, 2013.
14. *Answer Set Programming using Gringo/Clasp*. Oral. At the Keeping Current Seminar, Department of Computer Science, University of Kentucky, 2011.

Honors and Awards

Verizon Fellowship	Fall 2015 - Spring 2016
Graduate Teaching Assistantship	Fall 2014 - Spring 2015, Fall 2012 - Spring 2013
AAAI-15 Student Volunteer and Scholarship Award	Jan. 2015
International Student Tuition Scholarship	Jan. 2015
Nominee of the Dissertation Year Fellowship	Dec. 2014
Harrison D. Brailsford Graduate Scholarship	Oct. 2014
Kentucky Opportunity Fellowship Awards	July 2013 - June 2014
Nominee of the ACM Award for Outstanding Teaching Assistant	2013
NSF Student Travel Award	Aug. 2013
IJCAI-13 Travel Grant Award	Aug. 2013
Graduate Research Assistantship	Fall 2010 - Spring 2013
Daniel R. Reedy Quality Achievement Fellowship	Aug. 2010 - May 2013
UK Student Travel Funding Awards	2013 (IJCAI, ADT), 2014 (AAAI)
Chinese National Endeavor Scholarship	Fall 2008
Outstanding Student Scholarships	Fall 2006 - Spring 2010

Technical Skills

Programming languages: C/C++, C#, Java, Matlab, Python, Perl, SQL, PHP, HTML

System and software: Linux/Macintosh/Windows, Qt, Git, Answer Set Programming tools, Microsoft Visual Studio, Eclipse, MySql, Microsoft SQL Server

References

Dr. Miroslaw Truszczynski, Professor
Department of Computer Science, University of Kentucky
329 Rose Street, Lexington, KY 40506 USA
Office Phone: (859) 257-6738
Email: mirek@cs.engr.uky.edu

Dr. Christian Fritz, Area Manager
System Sciences Lab, Palo Alto Research Center
3333 Coyote Hill Road, Palo Alto, CA 94304 USA
Office Phone: (650) 812-4876
Email: cfritz@parc.com

Dr. Victor Marek, Professor

Department of Computer Science, University of Kentucky
329 Rose Street, Lexington, KY 40506 USA
Office Phone: (859) 257-3496
Email: marek@cs.uky.edu

Dr. Yi Pike, Lecturer
Department of Computer Science, University of Kentucky
329 Rose Street, Lexington, KY 40506 USA
Office Phone: (859) 218-5769
Email: yipike@cs.uky.edu