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

PREFERENCE TREES OVER COMBINATORIAL DOMAINS

By

Xudong Liu

Director of Dissertation: Miroslaw Truszczynski

Director of Graduate Studies: Miroslaw Truszczynski

Date: January 24, 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Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only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

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 an essential component in areas such as constraint satisfaction, decision making, and social choice theory. I focus on problems in preference representation, reasoning and learning.

Preferences can be represented in a quantitative or 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, however, people often cannot express their preferences directly and accurately as value functions [31].

Assume an agent is given three flavors of ice-cream *strawberry*, *chocolate* and *vanilla*, and 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:

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

This function shows that the favorite alternative to the agent is *chocolate* with 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 small pieces in a qualitative way, e.g., to specify binary preference relations. Thus, the same agent could rank the flavors as the following set of binary comparisons:

$$\{chocolate \succ vanilla, chocolate \succ strawberry, strawberry \succ vanilla\}.$$

Note that one can formulate the qualitative preferences from the value function, but not vice versa.

Several preference formalisms in both categories have been developed. Numeric preference systems include fuzzy constraint satisfaction [71] [78], generalized additive independence networks (GAI nets) [10, 51], conditionally utility independence networks (CUI nets) [35], and expected utility networks (EU nets) [66], while qualitative preference paradigms proposed in AI community include penalty logic, possibilistic logic, conditional preference networks (*CP-nets*) [53], conditional preference networks with tradeoffs (*TCP-nets*) [20], lexicographic preference trees (*LP trees*) [18], conditional preference theories (*CP theories*) [82], and answer set optimization theories (*ASO theories*) [25, 24].

Once we fix a preference formalism, say $\mathcal{F}$, in which preferences of agents are specified, learning preferences expressed in $\mathcal{F}$ from agents becomes a problem that has drawn attention from many AI researchers. Different techniques have been proposed to learning preferences in $\mathcal{F}$ such as active learning (or preference elicitation) and passive learning [43]. In the process of active learning, the algorithm asks the user for a pairwise preference relation between two given alternatives and constructs an instance of $\mathcal{F}$ as more pairwise relations are elicited. For passive learning, the algorithm assumes that a set of pairwise relations are somehow obtained and builds an instance of $\mathcal{F}$ with no more information from the user.

Provided with preferences learned and presented in a formalism, what can we say about the agent's preferences over outcomes? Reasoning problems can be classified based on the number of agents n from whom the preferences are elicited:

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 collective decision (e.g., a winning outcome or ranking) among the group of agents.

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

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

We are asked to compute the winner according to some voting rules. Plurality, veto and Borda are examples of commonly used voting rules. For each vote, plurality assigns score 1 to the top ranked alternative and 0 to the others, veto assigns score 0 to the bottom ranked alternative and 1 to the others, and Borda assigns score $m - i$ to the ith ranked alternative (m is the number of alternatives). Then the winner, thus, is the alternative with the highest score; in case of ties, we break them based on the order $a > b > c > d > e$. We compute the winner for P_X as follows.

1. Plurality: alternative a is the winner because the score of a is maximal.
2. Veto: since a, b and c are not vetoed but d and e are, a, b and c are tied and a wins.
3. Borda: the winner is a since its score, 31, is the highest.

While in the cases when the number of alternatives is small the preference-aggregating problems (winner determination, etc) have received wide attention in the literature, the problems concerning preferences over combinatorial domains have not been as much investigated. For example, a group of students plan on a joint vacation. They may consider issues $Time$, $Transportation$, $Destination$, $Duration$, etc.

Each issue has a domain of values that it can take, e.g., $Time$ may have three values $spring$, $summer$ and $fall$. There could be tens or hundreds of vacations even for a relatively small number of issues, and students will soon find it unlikely to enumerate all of them in a strict total order. Thus, applying voting rules directly is infeasible. Methods that consider individual issues one by one have been proposed and analyzed based on how well they approximate standard voting rules [39, 84].

Consequently, an expressive yet concise representation is needed in which preferences over combinatorial alternatives are specified in a compact way. Such models have been proposed in the literature and include conditional preference networks (CP-nets) [53], lexicographic preference trees (LP trees) [18] and answer set optimization theories (ASO theories) [25, 24]. Compact and expressive as they are, CP-nets are limited to describe preference relations under a *ceteris paribus* interpretation, and ASO theories apply a rather weak Pareto ordering to rank outcomes with respect to multiple objectives [25].

In our earlier work [59][1], we have considered the preference paradigm of LP trees. An LP tree is an intuitive and compact representation of preferences over alternatives from combinatorial domains. Informally, it exploits a natural way that humans apply to express preference information in the setting of combinatorial domains. Often a human agent would first consider the most important issue and specify her preference within the domain of that issue. Then, based on how the most important issue is evaluated in an alternative, the agent identifies the next important issue and specify her preference over values of that issue, and so on. This process ends up with a structured preference system that always induces a strict total order. In our paper, we obtained new complexity results for some preference aggregation problems (e.g. computing the winning alternative) according to positional scoring rules when votes are expressed as LP trees, and provided insight into effectiveness of computational

[1] A joint paper with my PhD advisor Dr. Miroslaw Truszczynski

tools on solving the problems of practical sizes.

For the future research, I will explore the relationship between different preference formalisms, especially between CP-nets and LP trees. Furthermore, I will extend the existing preference paradigms to allow more expressivity when it comes to representing complex preferences. In particular, I focus on a hybrid system where relations of subsystems are specified as a decision tree and preferences within each subsystem are expressed as partial preorders. We will also study preference learning techniques to elicit preference relations over combinatorial domains, aggregating partial LP trees, and manipulation problems in social choice when votes are LP trees.

The outline of the rest of this proposal is the following. In Section **??**, we considered related work that proposed approaches to preference representation and aggregation both in AI and social choice theory. In Section **??**, we focus on a specific preference formalism, LP trees, and present our own work on aggregating LP trees over combinatorial domains based on standard voting schemes, where both computational complexity and expirical results are shown and analyzed. Plan of the following research towards my PhD thesis is demonstrated in Section **??**.

Chapter 2 Technical Preliminaries

In this section I will give an overview of different mathematical and computational concepts that we will use throughout the rest of this document. First, since we consider preference relations that are modeled as binary relations, we recall binary relations, properties of binary relations and different preference relations satisfying some properties. Second, we review combinatorial domains because we focus on problems concerning preferences over combinatorial domains. Finally, in order to quantify how hard these problems are in a computational sense, we discuss concepts in computational complexity theory that are related to our research.

2.1 Relations

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,

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

Now let us review some of the important properties of binary relations.

Definition 2. Let R be a binary relation over a set O of objects ($R \subseteq O \times O$), we define the following properties of R:

1. reflexive: $\forall o \in O, oRo$.
2. irreflexive: $\forall o \in O, \neg oRo$.
3. total: $\forall o, o' \in O, oRo'$ or $o'Ro$.
4. transitive: $\forall o, o', o'' \in O$, if oRo' and $o'Ro''$, oRo''.
5. antisymmetric: $\forall o, o' \in O$, if oRo' and $o \neq o'$, $\neg o'Ro$.

For instance, assuming $\mathbb{N} = \{1, 2, \ldots\}$ is the set of positive integers, the at-most relation $\leq$ over $\mathbb{N}$ is reflexive, complete, 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 and total, a *partial order* if it is a partial preorder and antisymmetric, or a *total order* if it is a partial order and total.

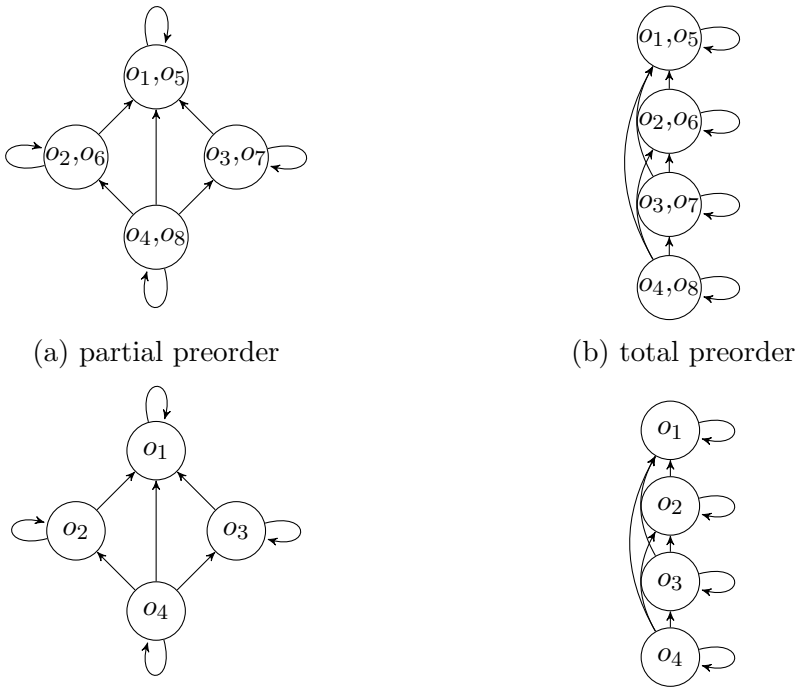
Preference relations can be modeled as binary relations as defined in Definition 3. Given two objects o and o', we sometimes need to specify that o' is at least as good as o or o' is strictly preferred over o. In some situations we need to speak about the fact that o' is as good as o', whereas in some other situations, due to some reasons (e.g., lack of informations about the two objects at hand), we just cannot determine which object is preferred over the other. Formally, we have the following definitions.

Definition 4. Let O be a set of objects, o and o' two objects in O. Let $\succeq$ be a preference relation that is a partial preorder [1] 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 indifferent from o, $o' \approx o$, if $o' \succeq o$ and $o \succeq o'$. Object o' is incomparable with o, $o' \sim o$, if $o' \not\succeq o$ and $o \not\succeq o'$.

Let us consider some examples of preference relations in Figure 2.1. We assume that a directed edge between two objects is from the less preferred object to the more preferred one. We also assume that relations obtained by transitivity are omitted. It is clear that the relation in Figure 2.1(a) is a partial order, and Figure 2.1(b) a total order.

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

[1] We only show the most general case where a preference relation is a partial preorder. Other cases will be defined accordingly.



(a) partial preorder (b) total preorder

(c) partial order (d) total order

Figure 2.1: Binary relations

As an example of relation extensions, we consider the partial order $\succeq$ in Figure 2.1(a). Since $o_2 \sim o_3$, 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 *maximal* if there exists $o' \in O$ such that $o' \succeq o$.

For instance, object o_1 is maximal in the total order shown in Figure 2.1(b).

2.2 Combinatorial Domains

We define the combinatorial domain as follows.

Definition 7. Let $\mathbf{V}$ be a set of variables (or features, attributes) $\{X_1, \ldots, X_p\}$, $\mathbf{D}$ a set of finite domains $\{Dom(X_1), \ldots, Dom(X_p)\}$ for each variable X_i. A *combinatorial*

domain is a triple $\langle \mathbf{V}, \mathbf{D}, \mathbf{T} \rangle$, where $\mathbf{T}$ is a set of tuples that are combinations of values from domains in $\mathbf{D}$ of variables in $\mathbf{V}$:

$$\mathbf{T} = \prod_{X_i \in \mathbf{V}} Dom(X_i).$$

An example of combinatorial domain is the space of *subsets* of $\mathbf{V}$. In this case, $\mathbf{D}$ is a set of binary domains, and a tuple $t \in \mathbf{T}$ is a combination of 0's and 1's. Note that $0_i \in t$ means that X_i is not in the corresponding subset, and that $1_i \in t$ means that X_i is. It is not hard to see that the set $\mathbf{T}$ of tuples is exactly the space $\mathbf{S}$ of subsets of $\mathbf{V}$.

Clearly, the size of $\mathbf{T}$ is exponential in p, the number of variables. Note that, even though sets $\mathbf{V}$ of binary variables are of main interest, the exponential growth of $|\mathbf{V}|$ makes it impossible for agents to directly assess their preferences. Also notice that, in many practical cases, hard constraints (e.g., propositional formulas) are identified to eliminate the infeasible outcomes.

2.3 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, I define classes of computational complexity used for such classification. We assume formiliarity with the concept of the *Turing machine* (TM). Details of it and other definitions discussed below can be found in complexity books by Garey and Johnson [44]; Lewis and Papadimitriou [58]; 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.

Definition 8. Let f be a TM. A *decision problem* (or a *language*) is a set L_f of strings ($L \subseteq \Sigma^*$) such that f accepts any string in L_f.

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).

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 $\mathbf{DTIME}(x)$ ($\mathbf{NTIME}(x)$) be a set of decision problems for which there exists a deterministic (nondeterministic, respectively) TM that solves any instance of the problems in time $c \cdot x$ for some constant $c > 0$. We now define the two classes as follows.

Definition 9. The class **P** (**NP**) contains 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 mostly believed that $\mathbf{P} \neq \mathbf{NP}$ [46].

One of the many complexity classes related to **P** and **NP** [46] is the class **coNP**, which contains problems that are complement 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 10. $\mathbf{coNP} = \{L : \overline{L} \in \mathbf{NP}\}$.

To characterize the most difficult problems in class C (**NP**, **coNP**, etc), it is helpful to introduce the definition of polynomial-time reducibility [46] and the idea of C-hardness.

Definition 11. 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* $f(x) \in L'$. We say that L' is C-*hard* if $L \leq_p L'$ for every L in class C.

Definition 12. Let C be a complexity class (**NP**, **coNP**, etc). A decision problem L' is C-*complete* if L' is in class C and L' is C-hard.

It is clear that, in order to prove C-completeness, one need to first show that $L' \in C$ (membership of class C), and then 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 algorithm that decides L.

Definition 13. Define the base case:

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathbf{P},$$

Then for $i \geq 0$, we denote by $\Delta_i^P + 1$ ($\Sigma_i^P + 1$) the set of decision problems solvable by a deterministic (nondeterministic, respectively) TM in polynomial time with an oracle for some Σ_i^P-complete problem. We denote by Π_{i+1}^P the set of decision problems that are complement of problems in $\Sigma_i^P + 1$.

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. We denote by **PH** the collection of all classes in the polynomial hierarchy.

The relation between any classes on the hierarchy is of interest. One may notice that the classes Σ_i^P and Π_i^P are complement, that is $\Sigma_i^P = \mathbf{co}\Pi_i^P$. Moreover, we have the relations of these classes as shown in Figure 2.2.

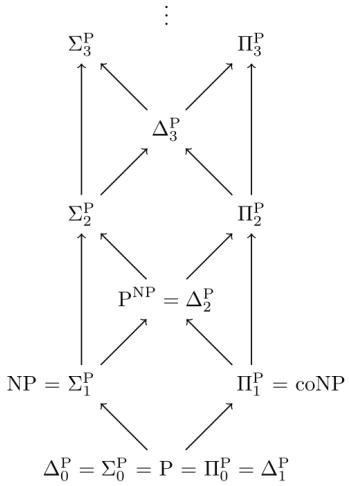


Figure 2.2: Polynomial hierarchy diagram

PSPACE

In this work, we consider yet another complexity class we will use later is **PSPACE** that concerns the complexity of space.

Definition 14. 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.

PH $\subseteq$ PSPACE.

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

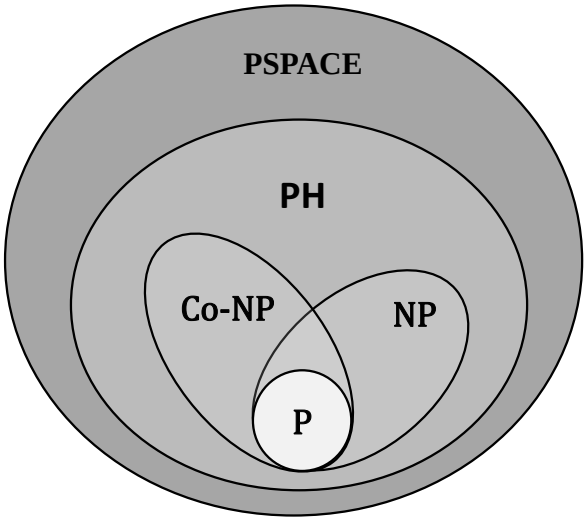


Figure 2.3: Computational complexity diagram

Chapter 3 Related Work

In this section, I will first revisit some preference systems in the literature that are designed to represent qualitative preferences. I will then give concepts in social choice that are often considered a mechanism combining individual preferences to reach a common decision.

3.1 Preferences Modeling and Reasoning

In the following, we recall two graphical preference formalisms, CP-nets and *Lexicographic Preference Trees* (LP trees), and two preferential logics, *Conditional Preference Theories* (CP Theories) and *Answer Set Optimization* (ASO). They are developed to provide concise and intuitive presentations of preferential information in combinatorial domains.

For either system, I will focus on three aspects [31]: (i) the language in which preferences are specified, (ii) the mathematical model captured by the system, and (iii) complexity of and algorithms for problems about the model (cf. the following definitions).

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

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 M of $\mathcal{C}$, decide whether M is an optimal outcome.

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

Conditional Preference Networks

The Language. CP-nets define preferential relations between outcomes based on the *ceteris paribus* semantics [19]. *Ceteris paribus* is latin for "everything else being equal."

Let $\mathbf{X}$ be a set of variables, we denote by $Asst(X)$ the set of all truth assignments to the variables in $\mathbf{X}$.

Definition 19. A set of variables $\mathbf{X}$ is preferentially independent of $\mathbf{Y} = \mathbf{V} - \mathbf{X}$ *iff* $\forall \mathbf{x}_1, \mathbf{x}_2 \in Asst(\mathbf{X})$ and $\forall \mathbf{y}_1, \mathbf{y}_2 \in Asst(\mathbf{Y})$,

$$\mathbf{x}_1\mathbf{y}_1 \succeq \mathbf{x}_2\mathbf{y}_1 \textit{ iff } \mathbf{x}_1\mathbf{y}_2 \succeq \mathbf{x}_2\mathbf{y}_2.$$

Definition 20. Let $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ be nonempty sets such that $\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z} = \mathbf{V}$. $\mathbf{X}$ is conditionally preferentially independent of $\mathbf{Y}$ given an assignment $\mathbf{z}$ to $\mathbf{Z}$ *iff* $\forall \mathbf{x}_1, \mathbf{x}_2 \in Asst(\mathbf{X})$ and $\forall \mathbf{y}_1, \mathbf{y}_2 \in Asst(\mathbf{Y})$,

$$\mathbf{x}_1\mathbf{y}_1 z \succeq \mathbf{x}_2\mathbf{y}_1 z \textit{ iff } \mathbf{x}_1\mathbf{y}_2 z \succeq \mathbf{x}_2\mathbf{y}_2 z.$$

Definition 21. For each variable X_i, $Pa(X_i)$ denotes its parent variables such that, given an assignment to $Pa(X_i)$, X_i is conditionally preferentially independent of $\mathbf{V} - (Pa(X_i) \cup \{X_i\})$.

Definition 22. Let $\mathbf{V}$ be a set of variables $\mathbf{V} = \{X_1, \ldots, X_n\}$. A CP-net over $\mathbf{V}$ is a tuple (H, Γ), where

1. H is a directed graph (V, E) specifying dependencies among variables, where for every $X_i \in V$ we have $Pa(X_i) = \{X_j : (X_j, X_i) \in E\}$,

2. Γ is a collection of conditional preference tables (CPTs) for all variables. A $CPT(X_i)$ consists of preference statements of the form

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

where $\mathbf{u} \in Asst(Pa(X_i))$ and $\succ^i_{\mathbf{u}}$ is a total order over $Dom(X_i)$ given u, everything else being equal.

The Model.

Definition 23. Let N be a CP-net over $\mathbf{V}$, $X_i \in \mathbf{V}$, and $\mathbf{U} = Pa(X_i)$. Given an assignment $\mathbf{u}$ to $\mathbf{U}$, a total order $\succ$ over $Asst(\mathbf{V})$ *satisfies* $\succ^i_{\mathbf{u}}$ if for all $\mathbf{y} \in Asst(\mathbf{Y})$ and all $x, x' \in Dom(X_i)$, $\mathbf{u}x\mathbf{y} \succ \mathbf{u}x'\mathbf{y}$ whenever $x \succ^i_{\mathbf{u}} x'$. $\succ$ *satisfies* $CPT(X_i)$ if it satisfies $\succ^i_{\mathbf{u}}$ for each $\mathbf{u} \in Asst(\mathbf{U})$. $\succ$ *satisfies* the CP-net N if it satisfies $CPT(X_i)$ for every X_i. A CP-net N is *consistent iff* there exists a total order $\succ$ that satisfies N.

Consider a CP-net N in Figure 3.1 over three binary variables [19]. In the dependency graph in Figure 3.1(a), an arrow points to a child variable from a parent variable. The preferences on B (C) depend upon the assignment made to A (B, respectively). This CP-net induces a partial order shown in Figure 3.1(b) where each arrow is from a less preferred outcome to a more preferred one. Since one pair of outcomes are incomparable, namely $\bar{a}\bar{b}\bar{c}$ and $a\bar{b}c$, there are two models, total orders, satisfying N as follows.

$$abc \succ ab\bar{c} \succ a\bar{b}\bar{c} \succ a\bar{b}c \succ \bar{a}\bar{b}\bar{c} \succ \bar{a}\bar{b}c \succ \bar{a}bc \succ \bar{a}b\bar{c},$$
$$abc \succ ab\bar{c} \succ a\bar{b}\bar{c} \succ \bar{a}\bar{b}\bar{c} \succ a\bar{b}c \succ \bar{a}\bar{b}c \succ \bar{a}bc \succ \bar{a}b\bar{c}.$$

Definition 24. CP-CONSISTENCY: given a CP-net N, decide whether N is consistent.

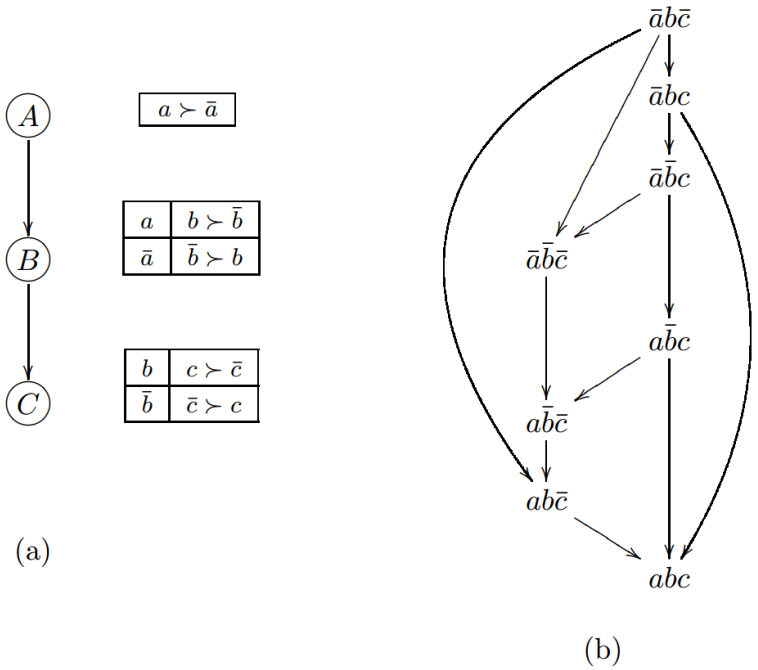


Figure 3.1: Acyclic CP-net

Answering the CP-CONSISTENCY problem depends upon whether its dependency graph are acyclic and how preference rules in CPTs are specified. Researchers in AI have been working on this problem and many results can be found in the literature. Boutilier, Brafman, Domshlak, Hoos and Poole [19] prove that every acyclic CP-net is consistent, whereas Goldsmith, Lang, Truszczynski and Wilson [50] show that the CP-CONSISTENCY problem is PSPACE-complete.

Problems and Complexity. The CP-DOMINANCE problem can be solved by a polynomial time algorithm for binary-valued tree-structured CP-nets, and that the problem is NP-complete for binary-valued CP-nets with specially structured dependency graphs (i.e., directed-path singly connectedness, max-δ-connectedness) [19]. However, it is NP-hard for general binary-valued acyclic CP-nets [19]. Furthermore, in the most general case when the dependency graph could be cyclic, this problem is proved PSPACE-complete even if the CP-nets are consistent[50].

The CP-OPTIMALITY-I problem, where an optimal outcome is an undominated one, is in general NP-complete even the dependency graph is cyclic [32].

The other two optimization problems, namely, the CP-OPTIMALITY-II and CP-OPTIMALITY-III problems, seem to have received less attention than other problems.

Lexicographic Preference Trees

The Language. Let us consider preferences over alternatives from combinatorial domains determined by a set $\mathcal{I} = \{X_1, X_2, \ldots, X_p\}$ of p binary *issues*, with each issue X_i having a binary domain $D(X_i) = \{0_i, 1_i\}$. The *combinatorial domain* in question is the set $\mathcal{X}(\mathcal{I}) = D(X_1) \times D(X_2) \times \ldots \times D(X_p)$. If $\mathcal{I}$ is implied by the context, we write $\mathcal{X}$ instead of $\mathcal{X}(\mathcal{I})$. For instance, let $\mathcal{I} = \{X_1, X_2, X_3\}$. A 3-tuple $(0_1, 1_2, 1_3)$ is an alternative from $\mathcal{X}(\mathcal{I})$, or simply, an alternative over $\mathcal{I}$. We often write it as $0_1 1_2 1_3$ or just as 011. It assigns 0 to X_1, 1 to X_2, and 1 to X_3. Clearly, the cardinality of $\mathcal{X}(\mathcal{I})$, which we denote by m throughout the paper, is 2^p.

A lexicographic preference tree (*LP tree*) T over a set $\mathcal{I}$ of p binary issues $X_1, \ldots, X_p$ is a *binary tree*. Each node t in T is labeled by an issue 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 issues labeling the ancestor nodes). We require that each issue appears exactly once on each path from the root to a leaf.

Intuitively, the issue labeling the root of an LP tree is of highest importance. Alternatives with the preferred value of that issue are preferred over alternatives 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 issue being the most important one. We note that the issues labeling the roots of the subtrees need not be the same (the relative importance of issues may depend on values for the issues labeling the nodes on the path to the root).

The precise semantics of an LP tree T captures this intuition. Given an alternative $x_1x_2 \ldots x_p$, we find its preference ranking in T by traversing the tree from the root to a leaf. When at node t labeled with the issue 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 explicitly represented level "splitting" into two of these implicit nodes, each representing an alternative. Descending the tree given an alternative in the way described above takes us to an (implicit) node at the lowest level that represents precisely that alternative. The more to the left the node representing the alternative, 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. A group of friends in Lexington want to make vacation plans for the next year. Having brainstormed for a while, the group decided to focus on three binary issues. The *Time* (X_1) of the travel could be either *summer* (s or 1_1) or *winter* (w or 0_1), the *Destination* (X_2) could be either *Chicago* (c or 1_2) or *Miami* (m or 0_2) and the mode of *Transportation* (X_3) could be to *drive* (d or 1_3) or *fly* (f or 0_3).

Jane, a member of the group, prefers a summer trip to a winter trip, and this preference on the issue *Time* is the most important one. Then for a summer trip, the next most important issue is *Destination* and she prefers Chicago to Miami, and the least important issue is *Transportation*. Jane prefers driving to flying if they go to Chicago, and flying, otherwise. For a winter trip, the importance of the remaining two issues changes with *Transportation* being now more important than the destination – Jane does not like driving in the winter. As for the *Destination*, she prefers to go to Miami to avoid the cold weather. These preferences can be captured by the LP

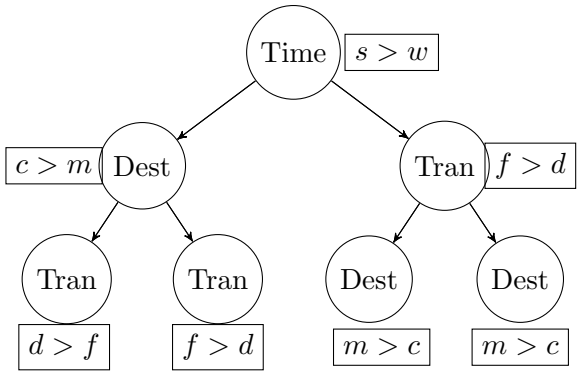


Figure 3.2: An LP tree T

tree T in Figure 6.1. We note that the trees ordering the vacation plans for summer and for winter are determined by trees with different assignments of issues to nodes. For instance, for the summer trips, the destination is the most important factor while for the winter trips the mode of transportation. The tree shows that the preferred vacation plan for Jane is to drive to Chicago in the summer and the next in order of preference is to fly to Chicago in the summer. The least preferred plan is to drive to Chicago in the winter.

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 issue), they can be collapsed to a single subtree, with the same assignment of issues to nodes. To retain preference information, at each node t' of the subtree we place a *conditional preference table*, and each preference in it specifies the preferred value for the issue labeling that node given the value of the issue labeling t. In the extreme case when for every node its two subtrees are identical, the tree can be collapsed to a path.

Since the preferred issue at a node depends on the values of issues above, the conditional preference table for the node t located at distance i from the root has possibly as many as 2^i rows (in general, 2^j rows, where j is the number of ancestor nodes with one child only), with each row specifying a combination of values for the

ancestor issues together with the preferred value for $Iss(t)$ given that combination. Thus, collapsing subtrees alone does not lead to a smaller representation size. However, it can be achieved if there are nodes whose preferred value depends only on a limited number of issues labeling their single-child ancestor nodes as in such cases the conditional preference table can be simplified.

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 issues 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. 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 6.1 can be represented more concisely as a (collapsed) CI-CP tree v in Figure 6.2. Nodes at depth one have their subtrees collapsed. In the tree in Figure 6.1, the subtrees of the node at depth 1 labeled *Tran* are not only identical but also have the same preference information at every node. Thus,

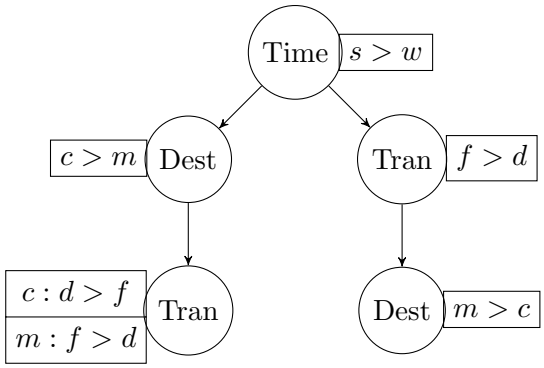


Figure 3.3: An CI-CP LP tree T

collapsing them does not incur growth in the size of the conditional preference table.

The Model. An LP tree consisting of p binary issues corresponds to a strict total order over 2^p alternatives. For the example in Figure 6.2, the total order induced by T is

$$scd \succ scf \succ smf \succ smd \succ wmf \succ wcf \succ wmd \succ wcd.$$

Problems and Complexity. An LP tree orders pairs of alternatives (M, M') by traversing down the tree and check the issues accordingly until an issue X is reach such that $M(X) \neq M'(X)$. M and M' are then ordered based on the preference information on X [18]. Thus, we have the following Theorem 1.

Theorem 1. *The LP-DOMINANCE problem can be solved in time linear in p.*

Proof. The linear time algorithm is shown in Algorithm 1.

□

Since an LP tree determines a linear order over alternatives, an optimal alternative always exists and the *LP*-OPTIMALITY-I problem is trivial. And it can be solved in time linear in p. Similarly, the *LP*-OPTIMALITY-II and *LP*-OPTIMALITY-III problems are easy to solve.

Instead of the problems of reasoning about a single LP tree, recently researchers have studied the problem of aggregating LP trees from multiple agents to facilitate

Algorithm 1: Solving the *LP*-DOMINANCE problem

Input: an LP tree T, two alternatives M and M' ($M \neq M'$)
Output: *true* if $M' \succ_T M$; *false*, otherwise

```
1  Let T* = T;
2  for i ← 1 to p do
3  |  Let X_j be the root of T* with preference x_j ≻ x̄_j;
4  |  if M'(X_j) > M(X_j) then
5  |  |  return true;
6  |  else if M'(X_j) < M(X_j) then
7  |  |  return false;
8  |  else
9  |  |  T* ← T*(x_j);
10 |  end
11 end
```

collaborative decision making. LP trees are aggregated according to some social choice scheme, such as issue-by-issue voting [39], sequential majority voting rule [84], positional scoring rules (e.g. Borda, k-Approval) [56, 59]. In Section **??**, I will provide detailed definitions of aggregating problems and their complexity according to positional scoring rules, as well as experimental analysis for two computational tools: Answer Set Programming (ASP) [48] and Weighted Partial Maximum Satisfiability (WPM) [69].

Conditional Preference Theories

A decade ago, Nic Wilson define a logic of conditional preferences, called Conditional Preference Theories (*CP theories*), that allows compact representation of certain kinds of conditional preference statements [82]. The logic generalizes both CP-net [82] and TCP-net [81]. Approximation of preference relations in CP theories, for which the DOMINANCE problem is in general very hard, has also been studied [83].

The Language. In the language of CP-nets, one can express her preferences over cars under the semantics of *ceteris paribus* as follows: *A Toyota car is preferred to a Ford car, everything else being equal.* Such semantics of CP-nets imposes limitation on

its expressivity for stronger preference statements, such as *A Toyota car is preferred to a Ford car, regardless of how they evaluate the other variables.* CP-nets cannot generally express such statements in a compact way [82], while CP theories can.

Definition 25. Let V be a set of variables. For each $X \in V$, let $\underline{X}$ be the set of values of X. For each $U \subset V$, let $\underline{U} = \Pi_{X \in U} \underline{X}$ be the set of assignments to U. A conditional preference statement φ is of the form

$$u : x > x'[W],$$

where $u \in \underline{U}$, $x, x' \in \underline{X}$ $(X \notin U)$, and $W \subseteq V - U - \{W\}$. A conditional preference theory Γ is a set of conditional preference statements.

The Model.

Definition 26. For $\varphi = u : x > x'[W]$, let T be a subset of V such that $T = V - \{U \cup \{X\} \cup W\}$. define φ^* the set of pairs of outcomes $\{(tuxw, tux'w') : t \in \underline{T}, w, w' \in \underline{W}\}$ induced by φ.

Each pair $(\alpha, \beta) \in \varphi^*$ represents the preference of α over β. Informally, φ expresses that, given u, x is preferred to x' regardless of the assignments w, w' to W, everything else (assignments t to T) being equal. Consider a set of cars over three binary variables *Make*, *Color* and *Category*, where $Dom(Make) = \{Toyota, Ford\}$, $Dom(Color) = \{black, white\}$, and $Dom(Category) = \{sedan, truck\}$. For a preference over cars " *For Toyota cars, a sedan is preferred to a truck, regardless of other variables*," we have the following preference statement φ:

$$Toyota : sedan > truck[\{Color\}].$$

We see that φ is a shorthand for the set φ^* of four binary relations it induces.

Definition 27. Let Γ be a conditional preference theory, define Γ^* the union of φ^* for all φ in Γ, that is, $\Gamma^* = \bigcup_{\varphi \in \Gamma} \varphi^*$. Assume preference relation is transitive, define $>_\Gamma$ the transitive closure of Γ^*.

We define models of Γ to be strict total orders on $\underline{V}$.

Definition 28. Let $>$ be a strict total order on $\underline{V}$. The relation $>$ is a model of ϕ, $> \models \varphi$ if and only if $> \supseteq \phi^*$. The relation $>$ is a model of Γ, $> \models \Gamma$, if and only if $> \supseteq \Gamma^*$. We say Γ is consistent if it has a model, that is, there exists a strict total order $>$ such that $> \models \Gamma$.

Indeed, we have $> \models \varphi$ if and only if $\alpha > \beta$ whenever $(\alpha, \beta) \in \varphi^*$. That is, for all $t \in \underline{T}$ and $w, w' \in \underline{W}$, $tuxw > tux'w'$.

Problems and Complexity. Let Γ be a CP theory, $G(\Gamma)$ a directed graph with edges representing dependency and importance relations. Assuming $G(\Gamma)$ is acyclic, sufficient and necessary conditions for a CP theory to be consistent have been characterized [82]: The CP theory Γ is consistent if and only if Γ is locally consistent. Under this assumption, the CPTh-CONSISTENCY problem can be solved efficiently. Moreover, computing a model of Γ and finding an optimal outcome can also be solved in polynomial time by constructing from Γ a partial conditional lexicographic order which can easily be extended to a total order [82]. Other sufficient conditions of consistency are discussed too [81].

In general, the CPTh-DOMINANCE problem is PSPACE-complete [83].

Answer Set Optimization

The formalism of Answer Set Optimization (ASO) was originally introduced by Brewka, Niemla and Truszczynski [26] and later enhanced by Brewka [24]. A declarative planning language, $\mathcal{PP}$, was introduced [75] and shown that any preference in $\mathcal{PP}$ can be embedded in ASO preferences [24].

In this work, we focus on the original framework [26] where the Pareto method is used to order outcomes. Other methods can be found in the latter paper [24]. In more recent works [36, 37], the Pareto-based ASO framework is generalized in the setting of qualitative optimization problems.

The Language.

Definition 29. 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 with hard constraints, 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 boolean combination built of atoms in A and α is a conjunction of literals of atoms in A.

The Model. A single ASO preference rule specifies a total preorder over outcomes, while, applying the Pareto ordering, a general ASO program with multiple preferences describe a partial preorder over the space of answer sets.

We say outcome o is irrelevant to preference rule r if $o \models (\neg\alpha) \vee (\neg\gamma_1 \wedge \ldots \wedge \gamma_k)$, that is, o does not satisfy α or o does not satisfy any of the Boolean combinations. As mentioned in the work by Brewka et al [26], outcomes irrelevant to r are considered as good as the best outcomes. Formally we define satisfaction degree of an answer set with respect to a preference rule.

Definition 30. Let o be an outcome generated by P_{gen}, r an ASO preference rule. The satisfaction degree of o on r, denoted $d_r(0)$, 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 31. Let (P_{gen}, P_{pref}) be an ASO theory, o and o' two outcomes. o' is weakly Pareto-preferred to o, $o' \succeq o$, if $\forall r \in P_{pref}, d_r(o') \leq d_r(o)$. o' is strictly Pareto-preferred to o, $o' \succ o$, if $o' \succeq o$ and $o \not\succeq o'$. o is optimal if there exists no outcome o'' such that $o'' \succ o$.

Consider an ASO theory $P = (P_{gen}, P_{pref})$, where

$$P_{gen} = \{\ 1\{a, \bar{a}\}1.\ \ 1\{b, \bar{b}\}1.\ \ 1\{c, \bar{c}\}1.\} \text{ and}$$
$$P_{pref} = \{a > \bar{a} \leftarrow b \vee c.\ \ b \wedge \bar{c} > \bar{b} \wedge c.\}.$$

An optimal outcome is $ab\bar{c}$ and it is the only optimal one.

Problems and Complexity. Brewka et al [26] 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.

Moreover, an extended paradigm of ranked ASO programs has been introduced in the same work [26]. Ranked ASO programs are ASO programs where rules in P_{pref} are given numeric values that represent different levels of importance of preference rules. Nonetheless, complexity results presented above stay unchanged.

3.2 Social Choice

The study of preference aggregation can be traced back to social choice theory, which dates from Condorcet's paradox of voting, noted by the *Marquis de Condorcet* in the 18th century, in which the winning ranking of alternatives could be cyclic even given acyclic individual votes [80]. 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 alternatives cannot meet some fairly desirable axioms. Built upon this Arrow's impossibility theorem, the Gibbard-Satterthwaite Theorem shows that any social choice function, again meeting some fair properties, is subject to manipulation [49, 72]. 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 [34].

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.

The field of computational social choice adds an algorithmic perspective from computer science to the formal approach of social choice theory [22].

Preference Aggregation

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

Social Welfare Functions.

Definition 32. Let $A = \{a_1, \ldots, a_m\}$ be a finite set of alternatives, $N = \{1, \ldots, n\}$ a finite set of agents (or voters). A preference relation (or a vote) v_i given by agent i is a strict 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 alternatives A, and $\mathcal{L}(A)^n$, the set of all preference profiles.

Definition 33. A social welfare function (SWF) is a function f:

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

We call the resulting relation $\succ$ the social preference relation.

If there are two alternatives 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 alternatives, the majority voting rule can lead to cycles of alternatives,

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

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

A general result, Arrow's theorem, shows that no social welfare functions satisfies some fairness conditions, defined as follows.

Definition 34. An *SWF* satisfies *Pareto efficiency* if

$$\forall i \in N, \forall a_j, a_k \in A, (a_j \succ_i a_k \Rightarrow a_j \succ a_k).$$

Let $N_{(a_j,a_k)}$ be the set of voters where a_j is preferred to a_k ($N_{(a_j,a_k)} = \{i : a_j \succ_i a_k\}$). An *SWF* satisfies *independence of irrelevant alternatives* if

$$\forall P, P' \in \mathcal{L}(A)^n, \forall a_j, a_k \in A, (N_{(a_j,a_k)} = N'_{(a_j,a_k)} \Rightarrow$$
$$a_j, a_k \text{ are ranked identically in } \succ \text{ and } \succ').$$

An *SWF* satisfies *non-dictatorship* if there is no agent i such that

$$\forall P \in \mathcal{L}(A)^n, \forall a_j, a_k \in A, (a_j \succ_i a_k \rightarrow a_j \succ a_k).$$

Theorem 2. (Arrow, 1951) *When there are at least three alternatives, There exists no SWF that simultaneously satisfies Pareto efficiency, independence of irrelevant alternatives and non-dictatorship.*

Social Choice Functions.

Definition 35. A social choice function (*SCF*) is a function f:

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

We call the resulting alternative (alternatives) an winner (co-winners, respectively).

We now discuss some of the desirable axioms of an *SCF* including resolution, anonymity and neutrality.

Definition 36. An *SCF* f is resolute if it always yields a unique winning alternative, that is,

$$\forall P \in \mathcal{L}(A)^n, |f(P)| = 1.$$

f is anonymous if names of the voters do not matter, that is, for every permutation π on voters,

$$f(v_1, \ldots, v_n) = f(v_{\pi(1)}, \ldots, v_{\pi(n)}).$$

f is neutral if names of the alternatives do not matter, that is, for every profile P and every permutation π on alternatives,

$$\pi(f(P)) = f(\pi(P)).$$

Surprisingly, these three fairness conditions cannot be satisfied simultaneously by any *SCF*. We have the following easy impossibility theorem.

Theorem 3. *A resolute SCF cannot satisfy both anonymity and neutrality.*

Proof. Consider the election with two votes over two alternatives a and b:

$$P = \{a \succ_1 b, b \succ_2 a\}.$$

We apply the resolute *SCF*: the majority rule with tie-breaking in favor of a. So a wins in P. Assuming anonymity, we still have a as the winner in profile P':

$$P' = \{a \succ_2 b, b \succ_1 a\}.$$

Let us assume neutrality also holds, b wins in P'. Contradiction!

□

Another desirable property of an *SCF* is that if the support of a winning alternative grows then it stays a winner in the new profile. This condition is called monotonicity. Formally, we define monotonicity as follows.

Definition 37. An *SCF* f satisfies *monotonicity* if

$$\forall a, b \in A, \forall P, P' \in \mathcal{L}(A)^n, (a \in f(P) \wedge N_{(a,b)} \subseteq N'_{(a,b)} \Rightarrow a \in f(P')).$$

For instance, the majority rule (plurality) does not satisfy monotonicity, but the Condorcet rule does. Another desired property is that every alternative has a chance to win, namely, surjectivity (or non-imposingness). More formally, an *SCF* f is surjective if for every alternative $a \in A$, there exists some profile P such that $a \in f(P)$. Regarding the aforementioned two fairness conditions, Muller and Satterthwaite [65] proved yet anther impossibility theorem, which is the counterpart of Arrow's theorem for *SCF*s.

Theorem 4. (Muller and Satterthwaite, 1977) *When there are at least three alternatives, any resolute SCF satisfying monotonicity and surjectivity is dictatorial.*

Voting Rules

We first define voting rules.

Definition 38. A voting rule r is a specific *SCF* proposed for practical use.

Positional Scoring Rules. For profiles over a set A of alternatives, 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 alternative 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 alternative 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 [59], 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 [21]. Under approval voting, each voter approves any number of alternatives and the winner, or co-winners, are those with the highest score. However, it is not considered as a voting rule as it does not accept preference profiles as input.

Condorcet Consistent Rules. A Condorcet winner is an alternative that wins every pairwise comparisons against each of the other alternatives. 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 [40]. Voting rules that are Condorcet consistent include the following, only to list a few [22].

1. Copeland's rule: An alternative 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 alternative a is the minimum number of votes against 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 alternatives in the profile of votes, and the top-ranked alternatives in these rankings are the co-winners.
4. Dodgson's rule: A winner is an alternative that can be made a Condorcet winner by a minimal number of swaps of adjacent alternatives 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 alternative, or in favor of a randomly picked alternative 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 39. 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 [49, 72], any reasonable voting rule is susceptible to manipulation given that the number of alternatives is at least three.

Theorem 5. (Gibbard, 1973; Satterthwaite, 1975). *When there are at least three alternatives, 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[22]. 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 [77].

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.

We first give a definition of the manipulation problem as follows.

Definition 40. Let r be a resolute voting rule. Given a profile P of n votes over m alternatives and an alternative c, the manipulation problem is to decide whether there exists a single vote v' such that $c = r(P \cup \{v'\})$.

The manipulation problem is proved to be NP-hard for several rules, such as second-order Copeland (SOC) [11], and single transferable vote (STV) [14]. A variant of the manipulation problem is called coalitional manipulation problem, where manipulative voters jointly make c a winner. Since the manipulation problem defined above is a special case of this variant, both SOC and STV are NP-hard. Additionally, the variant is also NP-hard for Copeland [38], maximin [85] and Borda [16, 28].

Chapter 4 Reasoning with Preference Trees over Combinatorial Domains

Abstract. 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

Preferences are essential in areas such as constraint satisfaction, decision making, multi-agent cooperation, Internet trading, and social choice. Consequently, preference representation languages and algorithms for reasoning about preferences have received much attention [53]. When there are only a few objects (or *outcomes*) to compare, it is both most direct and feasible to represent preference orders by their explicit enumerations. The situation changes when the domain of interest is *combinatorial*, that is, its elements are described in terms of combinations of values of *issues*, say $x_1, \ldots, x_n$ (also called *variables* or *attributes*), with each issue x_i assuming values from some set D_i — its *domain*.

Combinatorial domains appear commonly in applications. Since their size is exponential in the number of issues, they are often so large as to make explicit representations of preference orders impractical. Therefore, designing languages to represent preferences on elements from combinatorial domains in a concise and intuitive fash-

ion is important. Several such languages have been proposed including penalty and possibilistic logics [33], conditional preference networks (CP-nets) [19], lexicographic preference trees (LP-trees) [18], and answer-set optimization (ASO) programs [25].

In this paper, we focus our study on combinatorial domains with binary issues. We assume that each issue x has the domain $\{x, \neg x\}$ (we slightly abuse the notation here, overloading x to stand both for an issue and for one of the elements of its domain). Thus, outcomes in the combinatorial domain determined by the set $\mathcal{I} = \{x_1, \ldots, x_n\}$ of binary issues are simply complete and consistent sets of literals over $\mathcal{I}$. We denote the set of all such sets of literals by $CD(\mathcal{I})$. We typically view them as truth assignments (interpretations) of the propositional language over the vocabulary $\mathcal{I}$. This allows us to use propositional formulas over $\mathcal{I}$ as concise representations of sets of outcomes over $\mathcal{I}$. Namely, each formula φ represents the set of outcomes that satisfy φ (make φ true).

For example, 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 issues. 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 [33] and ASO programs [25]. 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.).

Trees such as the one in Figure 4.1 are called *preference trees*, or *P-trees*. They

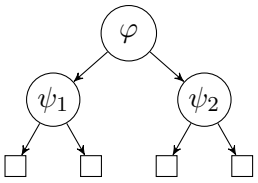


Figure 4.1: A preference tree

were introduced by Fraser [41, 42], 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 issue of compact representations received only an informal treatment by Fraser (P-trees in their full representation are often impractically large), and the algorithmic issues of reasoning with P-trees were also only touched upon.

In this paper, 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 [18]. 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.

Our paper 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 issues. 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 $M \in CD(\mathcal{I})$, we define the *leaf of M 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 $M \models \varphi$, we descend to the left child of t; otherwise, we descend to the right node of t. We denote the leaf of M in T by $l_T(M)$.

We use the concept of the leaf of an outcome M in a P-tree T to define a total preorder on $CD(\mathcal{I})$. Namely, for outcomes $M, M' \in CD(\mathcal{I})$, we set $M \succeq_T M'$, M is *preferred* to M', if $l_T(M) \succeq_T l_T(M')$, and $M \succ_T M'$, M is *strictly preferred* to M', if $l_T(M) \succ_T l_T(M')$. (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 M is *equivalent* to M', $M \approx_T M'$, if $l_T(M) = l_T(M')$. Finally, M is *optimal* if there exists no M' such that $M' \succ_T M$.

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, $M = \neg x_1 \neg x_2 \neg x_3 x_4$ and $M' = x_1 x_2 x_3 \neg x_4$, we walk down the tree and find that $l_T(M) = l_1$ and $l_T(M') = l_2$. Thus, we have $M \succ_T M'$

(a) Full (b) Compact

Figure 4.2: P-trees on vacations

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 issues, 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

$\succeq$. $\square$

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 issues, 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.

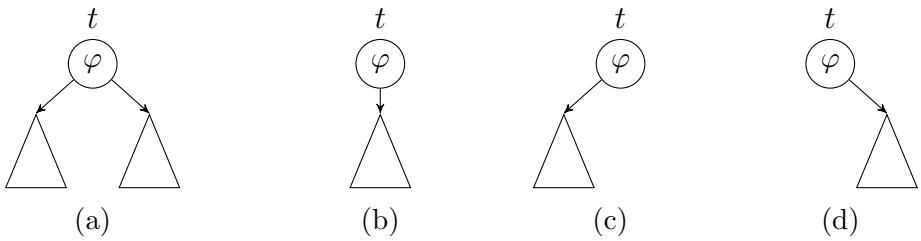


Figure 4.3: Compact P-trees

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.

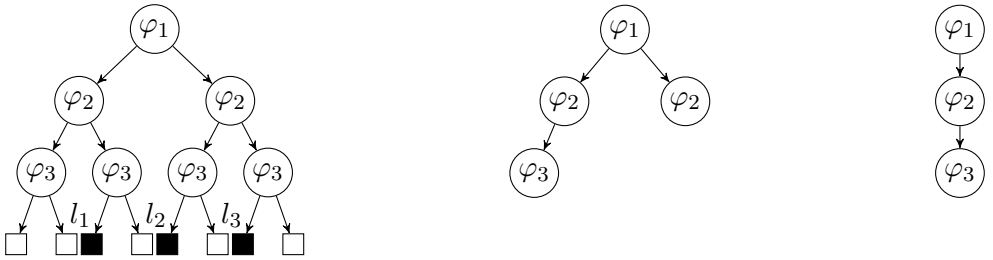


Figure 4.4: P-trees with empty leaves

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 [18, 57, 59]. In fact, LP-trees over a set $\mathcal{I} = \{x_1, \ldots, x_n\}$ of issues 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.

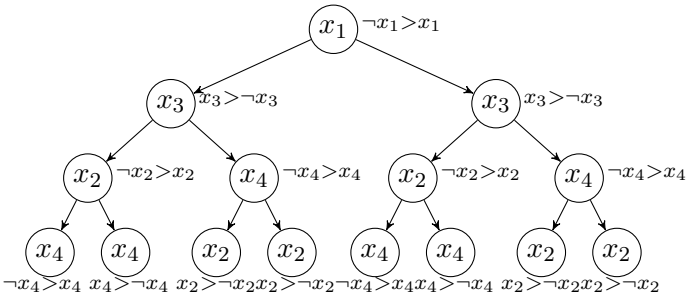
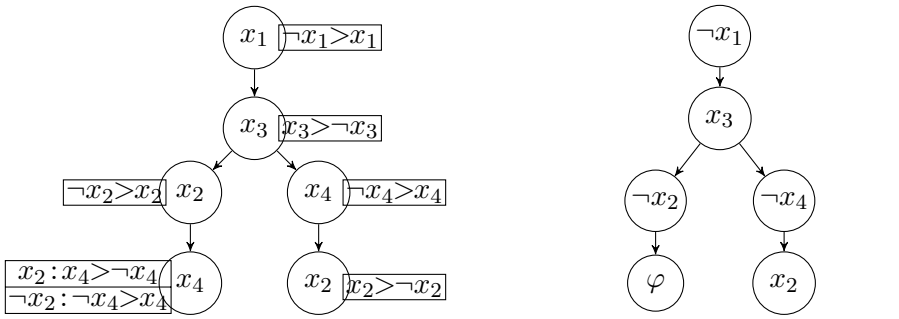


Figure 4.5: A full LP-tree on vacations

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 issues 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 [25] 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.[1]

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:

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

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

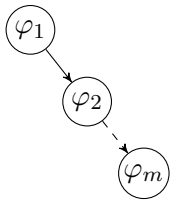


Figure 4.7: A P-tree T_r (T_P)

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 6. *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.

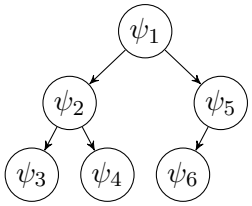


Figure 4.8: T_r^b when $m = 6$

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 6 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 [25], aggregated by the Pareto method.

We first show how Pareto method is used to order outcomes with regard to a set of *unranked* ASO-rules. Let M and M' be two outcomes. Given a set P of unranked ASO-rules, M is weakly preferred to M' with respect to P, $M \succeq_P^u M'$, if $SD_r(M) \leq SD_r(M')$ for every $r \in P$. Moreover, M is strictly preferred to M', $M \succ_P^u M'$, if $M \succeq_P^u M'$ and $SD_r(M) < SD_r(M')$ for some $r \in P$, and M is

equivalent to M', $M \approx_P^u M'$, if $SD_r(M) = SD_r(M')$ for every $r \in P$.

In general, the resulting preference relation is not total. However, by ranking rules according to their importance in some cases, total preorders can be obtained. Let us assume $P = \{P_1, \ldots, P_g\}$ 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 $M \succeq_P^{rk} M'$ w.r.t P if for every i, $1 \leq i \leq g$, $M \approx_{P_i}^u M'$, or if there exists a rank i such that $M \approx_{P_j}^u M'$ for every j, $j < i$, and $M \succ_{P_i}^u M'$.

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 7. *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'*

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

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. 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 Boolean 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 importance level, 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 outcome M with regard to preference pair (ϕ, a), $TD_{(\phi,a)}(M)$, is defined by

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

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

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

The larger $TD_\Pi(M)$, the more preferred M is.

For example, for the domain of vacations, we might have the following set of preference pairs $\{(\neg x_1 \wedge x_3, 0.8), (x_2 \wedge x_4, 0.5)\}$. According to the possibilistic logic interpretation, vacations satisfying both preferences are the most preferred, those satisfying $\neg x_1 \wedge x_3$ but falsifying $x_2 \wedge x_4$ are the next preferred, and those falsifying $\neg x_1 \wedge x_3$ are the worst.

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 8. *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'

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

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

Definition 42. Optimality-testing (OptTest): given a P-tree T and an outcome M of T, decide whether M is optimal.

Definition 43. 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 9. *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 10. *The* OptTest *problem is coNP-complete.*

Figure 4.9: The P-tree T_Φ

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 9). NP-hardness follows from a polynomial time reduction from SAT [45]. 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 issues, 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 11. *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[2] [55] 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. □

[2]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*.)

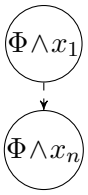


Figure 4.10: The P-tree T_Φ

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 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 Conclusion and Future Work

We investigated the qualitative preference representation language of *preference trees*, or *P-trees*. This language was introduced in early 1990s (cf. [41, 42]), but have not received a substantial attention as a formalism for preference representation in AI. We studied formally the issue 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 issues, 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 issues.

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 over Combinatorial Domains

Abstract. 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

Representing and reasoning about preferences are fundamental to decision making and so, of significant interest to artificial intelligence. When the choice is among a few *outcomes* (*alternatives*), representations in terms of explicit enumerations of the preference order are feasible and lend themselves well to formal analysis. However, in many applications outcomes come from *combinatorial domains*. That is, outcomes are described as tuples of values of *issues* (also referred to as *variables* or *attributes*), say $X_1, \ldots, X_p$, with each issue X_i assuming values from some set D_i — its *domain*. Because of the combinatorial size of the space of such outcomes, explicit enumerations of their elements are impractical. Instead, we resort to formalisms supporting intuitive and, ideally, concise *implicit* descriptions of the order. The language of CP-nets [19] is a prime example of such a formalism.

Recently, however, 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 [73, 30, 86, **?**]. The formalism of complete *lexicographic preference trees* (LP-trees) [18] 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) [23], 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. [18] considered learning of LP-trees, and Bräuning and Eyke [23] 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.

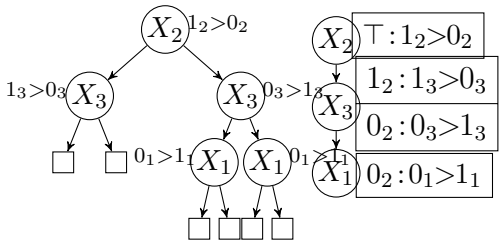
To illustrate the notions just introduced, we consider preference orderings of dinner options over four binary issues. The *appetizer* (X_1) can be either *salad* (0_1) or *soup* (1_1). The *main course* (X_2) is either *beef* (0_2) or *fish* (1_2). The *drink* (X_3) can be *beer* (0_3) or (white) *wine* (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 5.1a. 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).

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.

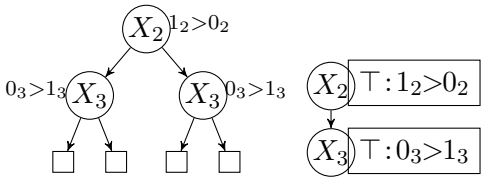
Let $R \subseteq \mathcal{I}$ be the set of issues that appear in a PLP-tree T. We say that T is *collapsible* if there is a permutation $\hat{R}$ of elements in R such that for every path in T from the root to a leaf, issues that label nodes on that path appear in the same order in which they appear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



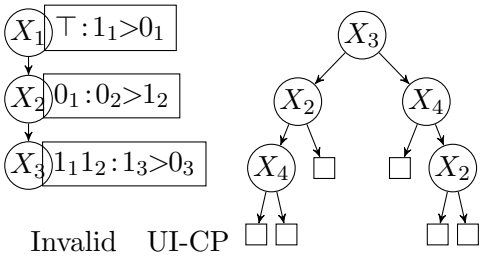
(a) Collapsible PLP-tree

(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

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 5.1a is collapsible, and can be represented compactly as a single-path tree with nodes labeled by issues in the permutation and PCPTs (cf. Figure 5.1b). 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 5.1b.

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 5.1d is the collapsed representation of the collapsible tree in Figure 5.1c.

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.

We now show the algorithm to turn a collapsible PLP-tree T into its compact *UI-CP* representation T'. We would first eliminate the leaves. Since subtrees in T are collapsible as well, we traverse and collapse subtrees of inner roots in a post-order

manner. When combining preferences for corresponding nodes, we will introduce the new condition (upon the root of the subtrees collapsed) unless the CPTs of these two nodes are the same. See example.

As an aside, we note that not every path of nodes labeled with issues and CPTs is a valid *UI* tree. Before define validity of such paths, we examine two properties of CPT's: completeness and consistency.

Let CPT(X_i) be the CPT of issue X_i consisting of conditional preference rules r_i's of form $cond_i :>_i$. We define the set of X_i's parent issues $Pa(X_i) = \bigcup_{r_i \in CPT(X_i)} Iss(cond_i)$, where $Iss(cond_i)$ is the set of issues that appear in $cond_i$, and $Iss(\top) = \emptyset$. The CPT of issue X_i is *complete* if for every assignment $v \in Asst(Pa(X_i))$ there exists a preference rule r_i such that $v \models cond_i$. CPT's that are not complete are *partial*. Another concept before validity is the consistency of a CPT. A CPT is *inconsistent* if there exist an outcome and two conditional preference rules in this PCPT such that the outcome satisfies both conditions but the preference orderings are different. A CPT is *consistent* if it is not inconsistent.

We obtain the following complexity results for checking completeness and consistency of a CPT. The problem of deciding completeness of a CPT is coNP-complete (cf. Theorem 12), and checking consistency of a CPT can be done in polynomial time.

Theorem 12. *Let PCPT(X) in a UI-CP PLP-tree T be the PCPT of issue X consisting of conditional preference rules r's of form cond :>. The problem to decide if PCPT(X) is complete is coNP-complete.*

Proof. (Membership) The complement of the problem is to decide if PCPT(X) is not complete, that is, to decide whether there exists an assignment $w \in Asst(Pa(X))$ for every preference rule r such that $w \not\models cond$. This complement problem is clearly in NP.

(Hardness) We show the hardness of the above complement problem by a reduction from SAT. Given an instance of SAT $\Phi = \{C_1, \ldots, C_g\}$ over Boolean variables $X_1, \ldots, X_n$, we construct a PCPT as follows.

1. Introduce a new variable X_{n+1} for which we build the PCPT.

2. For each clause $C_i \in \Phi$, add into PCPT(X_{n+1}) a conditional preference rule r_i of form $cond_i :>_i$, where $cond_i = \neg C_i$ and $>$ is set arbitrarily.

Clearly, PCPT(X_{n+1}) is not complete if and only if Φ is satisfiable. □

Now, let T be a path of $n < p$ nodes labeled with issues and consistent CPT's. For simplicity, we assume the order of issues is $X_1, \ldots, X_n$. We say that T is a *valid UI* tree if there exists a full PLP-tree that can be collapsed into T using the algorithm described above.

An example is given in Figure 5.1b. The condition characterizing paths with nodes labeled with issues and PCPTs that are valid *UI* trees is the following.

Let v be a partial interpretation over $\mathcal{I}$, and $S \subseteq \mathcal{I}$ a subset of $\mathcal{I}$. We denote with $v|_S$ the *projection* of v onto S. In particular, if $v|_\emptyset = \bot$. We now show the conditions for a path to be a valid UI-CP PLP-trees in the following theorem.

Theorem 13. *Let T be a path of nodes labeled with issues and consistent PCPT's. We have that T is valid if and only if for every node $t \in T$ labeled by X_i and CPT(X_i), for every preference rule r_i in CPT(X_i), for every parent issue $X_j \in Iss(cond_i)$ with CPT(X_j), there exists a preference rule r_j in PARTIAL CPT(X_j) such that*

$$cond_i|_{Iss(cond_j)} \models cond_j.$$

Proof. Prove by induction. Let N, $1 \leq N \leq p$, be the number of issues in T. It is clear when N is one. Now assuming the theorem holds for UI-CP trees of $1 \leq n < p$ issues, we show it also holds for an arbitrary UI-CP tree T with $N = n + 1$.

($\Leftarrow$) Let T_n be the tree obtained from T by removing the bottom node together with its issue and PCPT labels. Let T'_n be the full tree expanded from T_n. Then, we know that the orderings of issues given by the paths in T'_n are consistent. $\square$

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 44. 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 45. 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 46. 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 2. 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.

Algorithm 2: 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

1 $T \leftarrow$ empty sequence;
2 **while** $\mathcal{E}^{\succ} \neq \emptyset$ **do**
3 | Construct $AI(\mathcal{E}, S)$;
4 | **if** $AI(\mathcal{E}, S) = \emptyset$ **then**
5 | | **return** FAILURE;
6 | **end**
7 | $X_l \leftarrow$ an element from $AI(\mathcal{E}, S)$;
8 | Construct $PCPT(X_l)$;
9 | $T \leftarrow T, (X_l, PCPT(X_l))$;
10 | $\mathcal{E} \leftarrow \mathcal{E} \backslash \{e \in \mathcal{E}^{\succ} : e \text{ is decided on } X_l\}$;
11 | $S \leftarrow S \backslash \{X_l\}$;
12 **end**
13 **return** T;

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).

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. Let T be $X_{j_1} \triangleright \ldots \triangleright X_{j_k}$ $(k \leq p)$.

(1). If $X \notin \{T\}$, we have $T' = X \triangleright X_{j_1} \triangleright \ldots \triangleright X_{j_k}$. For any $e = (\alpha, \beta, 0) \in \mathcal{E}^{\approx}$, we know $\alpha(X) = \beta(X)$ since $X \in AI(\mathcal{E}, S)$, and $\alpha(X_{j_i}) = \beta(X_{j_i})$ for all $1 \leq i \leq k$ because $\alpha \approx_T \beta$. Thus, we have $\alpha \approx_{T'} \beta$. Let $e' = (\alpha', \beta', 1) \in \mathcal{E}^{\succ}$ be any strict example decided on issue X_{j_l} in T. That is, we know $\alpha'(X_{j_l}) = 1$ and $\beta'(X_{j_l}) = 0$, and $\alpha(X_{j_i}) = \beta(X_{j_i})$ for all $1 \leq i < l$. By the definition of $AI(\mathcal{E}, S)$, we have

$\alpha'(X) \geq \beta'(X)$. If $\alpha'(X) = \beta'(X)$, example e' is still decided on issue X_{j_l} in T'. Otherwise, we have $\alpha'(X) = 1$ and $\beta'(X) = 0$, and e' is decided on issue X in T'. Hence, we have that $\alpha \succ_{T'} \beta$.

(2). If $X \in \{T\}$, let j_r, $1 \leq r \leq k$, be the index such that $X = X_{j_r}$. Now we have $T' = X \triangleright X_{j_1} \triangleright \ldots \triangleright X_{j_{r-1}} \triangleright X_{j_{r+1}} \triangleright \ldots \triangleright X_{j_k}$. Tree T' is consistent with $\mathcal{E}^{\approx}$, done in case (1). Let $\mathcal{E}^{\succ}_{j_l}$ be the set of examples in $\mathcal{E}^{\succ}$ that are decided on issue X_{j_l} in T. We have that examples in every $\mathcal{E}^{\succ}_{j_l}$, $1 \leq l \leq r-1$, are decided in T', again done in case (1). For $l = r$, examples in $\mathcal{E}^{\succ}_{j_l}$ are decided on X in T'. For $r+1 \leq l \leq k$, examples in every $\mathcal{E}^{\succ}_{j_l}$ are decided on X_{j_l}.

Therefore, tree T' is consistent with $\mathcal{E}$. □

We now specialize Algorithm 2 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 2.

Theorem 14. *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 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 show that the problem is in fact NP-complete.

We first show that the Weak Consistency problem for a special class of acyclic CP-nets is NP-complete. A CP-net $N = (G, CPT)$ contains a directed graph G over $\mathcal{I}$ determining the dependencies between issues, and a set CPT of conditional preference tables $CPT(X_i)$ for every $X_i \in \mathcal{I}$. For instance, if X_i has r incoming edges in G, then $CPT(X_i)$ is a table of 2^r conditional preference orderings on D_i for all assignments to these r parent nodes. We say a CP-net $N = (G, CPT)$ is *total acyclic* if G is acyclic and $G = G^+$, where G^+ is the transitive closure of G.

Let N be a total acyclic CP-net, $\succ_N$ be the preference order induced by N, and $e = (\alpha, \beta, 1)$ be a strict example. We say that N is consistent with e if there is a completion $\succ$ of $\succ_N$ such that $\alpha \succ \beta$.

Definition 47. Let N be a total acyclic CP-net over $\mathcal{I}$, and $\mathcal{E}^{\succ}$ be a set of strict examples. We say that N is weakly consistent with $\mathcal{E}^{\succ}$ if N is consistent with every example in $\mathcal{E}^{\succ}$.

Theorem 15. *Let $\mathcal{E}^{\succ}$ be a set of strict examples. The weak consistency problem, to decide whether there exists a total acyclic CP-net N that is weakly consistent with $\mathcal{E}^{\succ}$, is NP-complete.*

Proof. For the membership, we can guess a directed dependency graph G over $\mathcal{I}$ and verify that G is total acyclic in polynomial time. We can show that if there is a total acyclic CP-net $N = (G, CPT)$ weakly consistent with $\mathcal{E}^{\succ}$ (*a priori*, the tables CPT may have size exponential in the size of $\mathcal{E}^{\succ}$), then another such CP-net of polynomial size exists too. The hardness comes from the fact that a special case, the *weak separability problem* [**?**], is NP-complete. □

Lemma 2. *Let $N = (G, CPT)$ be a total acyclic CP-net, and $e = (\alpha, \beta, 1)$ be a strict example. We have that N is consistent with e if and only if there exists an issue $X_i \in \mathcal{I}$ such that $\alpha(U_i) = \beta(U_i)$ and $\alpha(X_i) >_i \beta(X_i)$ given $\alpha(U_i)$ (or $\beta(U_i)$), where U_i is the set of parent issues of X_i in G, and $>_i$ is the total order over D_i.*

Proof. ($\Leftarrow$) It is proved in the setting of *ordering queries* [19].

($\Rightarrow$) Assume for every issue $X_j \in \mathcal{I}$ we have $\alpha(U_j) = \beta(U_j)$ and $\alpha(X_j) \not>_j \beta(X_j)$ given $\alpha(U_j)$ (or $\beta(U_j)$), we need to show that $\beta \succ_N \alpha$, that is, there is a sequence of improving flips from α to β in the preference graph $\succ_N$.

Case 1: Assume for every issue $X_j \in \mathcal{I}$ we have $\alpha(U_j) = \beta(U_j)$ and $\alpha(X_j) = \beta(X_j)$ given $\alpha(U_j)$ (or $\beta(U_j)$). Then, it must be that $\alpha = \beta$, which is impossible.

Case 2: Assume for some issue X_k we have $\alpha(U_k) = \beta(U_k)$ and $\beta(X_k) >_k \alpha(X_k)$ given $\alpha(U_k)$ (or $\beta(U_k)$). Then, it must be that $X_k \in G$ has an out degree of 0. It is easy to see there is a sequence of improving flips from α to β in $\succ_N$. □

Theorem 16. *The* ConsLearn *problem is NP-complete for class UI-CP.*

Proof. Membership is clear. We now prove that the problem is NP-hard by showing a polynomial time reduction from the *weak consistency problem*. Let $\mathcal{E}^{\succ}$ be a set of

strict examples. We show that there exists a total acyclic CP-net N weakly consistent with $\mathcal{E}^{\succ}$ if and only if there exists a UI-CP PLP-tree T consistent with $\mathcal{E}^{\succ}$.

($\Rightarrow$) Assume N is weakly consistent with $\mathcal{E}^{\succ}$, we construct T, where the order of issues is a linear completion of G and the *PCPT*s are copies of *CPT*s in N for corresponding issues. Since $e = (\alpha, \beta, 1) \in \mathcal{E}^{\succ}$ is consistent with N, by Lemma 2, there is a preference rule $u :>_i \in CPT(X_i)$ such that $\alpha(U) = \beta(U) = u$ and $\alpha(X_i) >_i \beta(X_i)$. It is clear that e is decided on X_i in tree T.

($\Leftarrow$) Assume T is consistent with $\mathcal{E}^{\succ}$, we build N, where G is constructed in a way that G is consistent with the dependencies between issues in T, and *CPT*s are copies of *PCPT*s in T for same issues. If $e = (\alpha, \beta, 1) \in \mathcal{E}^{\succ}$ is decided on X_i in T, on preference rule $u :>_i \in CPT(X_i)$ in N we have $\alpha(U) = \beta(U) = u$ and $\alpha(X_i) >_i \beta(X_i)$. Thus, by Lemma 2, N is weakly consistent with all e in $\mathcal{E}^{\succ}$ and hence $\mathcal{E}^{\succ}$. □

The SmallLearn Problem

Algorithm 2 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 17. *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 [45]. 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 18. *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 2. 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 2 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 19. *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 20. *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 [73], adapted to the case of *UI* PLP-trees. We now show hardness for the class *UI-CP* on top of the construction of examples shown by Schmitt and Martignon [73].

We first recall the *vertex cover problem* (VCP): given an undirected graph $G = (V, E)$ with n vertices and g edges, and a positive integer $k \leq n$, the problem is to decide whether there is a vertex cover of cardinality at most k, i.e., a subset $V' \subseteq V$ with $|V'| \leq k$ such that for every edge $(v_i, v_j) \in E$ at least one of v_i and v_j belongs to V'. Given an instance of VCP, we construct an instance of our problem as follows.
1. $\mathcal{I} = \{X_i : v_i \in V\} \cup \{X_{n+1}\}$.
2. $\mathcal{E} = \{(\langle \mathbf{1}_i, 1\rangle, \langle \mathbf{1}, 0\rangle, 1) : i = 1, \ldots, n\} \cup \{(\langle \mathbf{1}, 0\rangle, \langle \mathbf{1}_{i,j}, 1\rangle, 1) : (v_i, v_j) \in E\}$, where $\mathbf{1}$ and $\mathbf{1}_i$ ($\mathbf{1}_{i,j}$) stand for the n-bit vector of all 1's, the n-bit vector of all 1's except for position i (respectively, positions i and j) where it has a 0.

3. We set $l = k$.

We show that graph G has a vertex cover of size at most k if and only if the set $\mathcal{E}$ over $\mathcal{I}$ has a *UI-CP* PLP-tree that falsifies at most l examples in $\mathcal{E}$.

($\Rightarrow$) Assume G has a vertex cover V' of size k such that $V' = \{v_{i_1}, \ldots, v_{i_k}\}$. We build a *UI-FP* tree (a special *UI-CP* tree) $T = X_{i_1} \triangleright \ldots \triangleright X_{i_k} \triangleright X_{n+1}$ with preference $1 > 0$ on every issue. Since V' is a vertex cover, any edge example $(\langle \mathbf{1}, 0\rangle, \langle \mathbf{1}_{i,j}, 1\rangle, 1)$ and any vertex example $(\langle \mathbf{1}_i, 1\rangle, \langle \mathbf{1}, 0\rangle, 1)$ with $v_i \notin V'$ are satisfied by T, and any vertex example with $v_i \in V'$ is falsified by T. Thus, the set $\mathcal{E}$ has a *UI-CP* tree falsifying no more than $l = k$ examples.

($\Leftarrow$) Assume T is a *UI-CP* tree that falsifies at most l examples in $\mathcal{E}$. Build the set V' as follows.

(1). Let $v_i \in V'$ for every falsified vertex example $(\langle \mathbf{1}_i, 1\rangle, \langle \mathbf{1}, 0\rangle, 1)$.

(2). Let one of $v_i, v_j \in V'$ for every falsified edge example $(\langle \mathbf{1}, 0\rangle, \langle \mathbf{1}_{i,j}, 1\rangle, 1)$.

We assume that V' of size at most l is not a vertex cover. Then, there exists an edge $(v_i, v_j) \in E$ such that its edge example and two vertex examples are satisfied by T. Let T' be the full representation of T. Since the edge example $e = (\langle \mathbf{1}, 0\rangle, \langle \mathbf{1}_{i,j}, 1\rangle, 1)$ is satisfied by T', there exists a path in T' from root to a node labeled by the issue that decides e. This issue, say X, must be either X_i, X_j or X_{n+1}.

(1°). If e is decided on X_i, the preference on X_i must be $1_i > 0_i$. Then, the vertex example $(\langle \mathbf{1}_i, 1\rangle, \langle \mathbf{1}, 0\rangle, 1)$ is falsified by T', contradiction.

(2°). If e is decided on X_j, the preference on X_j must be $1_j > 0_j$. Then, the vertex example $(\langle \mathbf{1}_j, 1\rangle, \langle \mathbf{1}, 0\rangle, 1)$ is falsified by T', contradiction.

(3°). If e is decided on X_{n+1}, the preference on X_{n+1} must be $0_{n+1} > 1_{n+1}$. Then, both vertex examples are falsified by T', contradiction.

Thus, the set V' is a vertex cover. Therefore, the theorem holds. □

Corollary 21. *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 22. *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. [18] 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 3 adjusted for both classes.

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 $X \in AI(\mathcal{E}, S)$.*

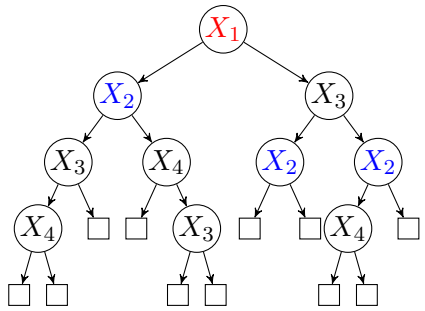
Algorithm 3: The recursive procedure *learnCI* that learns a CI PLP-tree

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

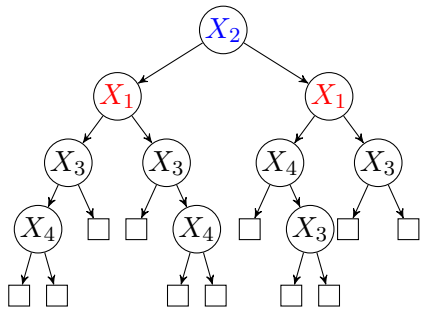
```
if E^≻ = ∅ then
    Label t as a leaf and return;
end
Construct AI(E, S);
if AI(E, S) = ∅ then
    return FAILURE and terminate;
end
Label t with tuple (X_l, x_l) where X_l is from AI(E, S), and x_l is the preferred
  value on X_l;
E ← E\{e ∈ E^≻ : e is decided on X_l};
S ← S\{X_l};
Create two edges u_l, u_r and two unlabeled nodes t_l, t_r such that u_l = ⟨t, t_l⟩ and
  u_r = ⟨t, t_r⟩;
E_l ← {e ∈ E : α_e(X_j) = β_e(X_j) = x_l};
E_r ← {e ∈ E : α_e(X_j) = β_e(X_j) = x̄_l};
learnCI(E_l, S, t_l);
learnCI(E_r, S, t_r);
```

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



(a) T



(b) T'

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

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

Proposition 3 clearly implies the correctness of Algorithm 3 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 23. *Let $\mathcal{E}$ be a set of examples over a set $\mathcal{I}$ of binary issues. Algorithm 3 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 17, whereas the hardness of the MaxLearn problem for *CI* cases follows from the proof by Schmitt and Martignon [73].

5.6 Conclusions and Future Work

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 Figure 5.3. 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.

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.

	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

Figure 5.3: Complexity results for passive learning problems

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 Aggregating Lexicographic Preference Trees over Combinatorial Domains

Abstract. 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 issues. 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.

6.1 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 *issues* (or attributes), and are specified by tuples of issue values. Spaces of such alternatives, often called *combinatorial domains*, are large. Indeed, the number of alternatives grows exponentially with the number of issues. 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 [29], possibilistic logic [33], conditional preference networks (CP nets) [19], 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-issue* aggregation addresses the computational hardness problem but often leads to results different from those obtained by applying common voting rules [39].

In this paper, we assume that votes are represented as *lexicographic preference trees*, or *LP trees*, for short [18], and that they are aggregated by some simple positional scoring rules such as Borda, k-approval and a refinement of the latter, (k, l)-

[1] Kaci [53] offers a comprehnsive discussion of preference formalisms.

approval. Given this setting, we study computing the best alternative, and the related 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 [57]. 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 [57]. 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, 67] and weighted partial maximum satisfiability (WPM-SAT) [6, 5], and apply to the encodings the ASP solvers *clingo* [47] and *clingcon* [68], 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* [74]. 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.

6.2 Technical Preliminaries

A *vote* over a set $\mathcal{X}$ of *alternatives* (or *outcomes*) is a *strict total* order $\succ$ on $\mathcal{X}$. In this work, we consider votes over alternatives from *combinatorial domains* determined by a set $\mathcal{I}$ of p binary *issues* $X_1, X_2, \ldots, X_p$, with each issue X_i having a binary domain $D(X_i) = \{0_i, 1_i\}$. The combinatorial domain in question is then the set $\mathcal{X}(\mathcal{I}) = D(X_1) \times D(X_2) \times \ldots \times D(X_p)$. If $\mathcal{I}$ is implied by the context, we write $\mathcal{X}$ instead of $\mathcal{X}(\mathcal{I})$. For instance, let $\mathcal{I} = \{X_1, X_2, X_3\}$. A 3-tuple $(0_1, 1_2, 1_3)$ is an alternative from $\mathcal{X}(\mathcal{I})$, or simply, an alternative over $\mathcal{I}$. We often write it as $0_1 1_2 1_3$ or just as 011. It assigns 0 to X_1, 1 to X_2, and 1 to X_3.

Clearly, the cardinality of $\mathcal{X}(\mathcal{I})$, which we denote by m throughout the paper, is 2^p. Thus, even for relatively small values of p, eliciting precise orders over all alternatives and representing them directly may be infeasible. Instead, in cases when votes have some structure, they often can be represented compactly by means of intuitive "preference expressions" in logical or graphical formalisms [19, 57, 53].

In this work we focus on one such formalism, the language of *lexicographic preference trees* or *LP trees*, for short [18]. An *LP tree* T over a set $\mathcal{I}$ of p binary issues $X_1, \ldots, X_p$ is a *binary tree*. Each node t in T is labeled by an issue 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 issues labeling the ancestor nodes). In addition, we require that each issue appears exactly once on each path from the root to a leaf. Figure 6.1(a) shows an example of an LP tree.

Intuitively, the issue labeling the root of an LP tree is of highest importance. Alternatives with the more preferred value of that issue are preferred over alternatives with the other (less preferred) value. The two subtrees further refine that ordering. The left subtree determines the ranking of alternatives in the more preferred "upper half" and the right subtree determines the ranking of alternatives in the less preferred "lower half." In each case, the same principle is used recursively, with the issue labeling the root of the subtree being the most important one. We note that the issues labeling the roots of the subtrees need not be the same (the relative importance of issues may depend on values for their "ancestor" issues labeling the nodes on the path to the root).

The precise semantics of an LP tree T is based on this intuition. To specify it, it is convenient to imagine that each node at the bottom of the tree has two children, the left one corresponding to the more preferred value of the issue labeling it, and the right one corresponding to the other, less preferred one. The nodes of that level are labeled from 1 to 2^p. To illustrate, in Figure 6.1(b) we show the tree from Figure 6.1(a) with the last level present.

Given an alternative $\bar{x} = x_1 x_2 \ldots x_p$, we find its preference rank in T by traversing the extended tree from the root to a leaf. When at node t labeled with the issue X_i, we follow down to the left subtree if x_i is preferred according to the preference information at t. Otherwise, we follow down to the right subtree. In this way, we end up in a unique leaf of the extended tree. Let us denote its label by $l(\bar{x})$. We take $l(\bar{x})$ as the *preference rank* of $\bar{x}$ and say that an alternative $\bar{x}$ is *preferred* (in T) to an alternative $\bar{y}$ precisely when $l(\bar{x}) \leq l(\bar{y})$ (informally, alternatives with lower ranks are preferred to those that are ranked higher). In this way, T defines a vote (a preference order).

To illustrate these concepts, let us consider an example. A group of friends want to plan a vacation trip for the next year. Having brainstormed for a while, the group

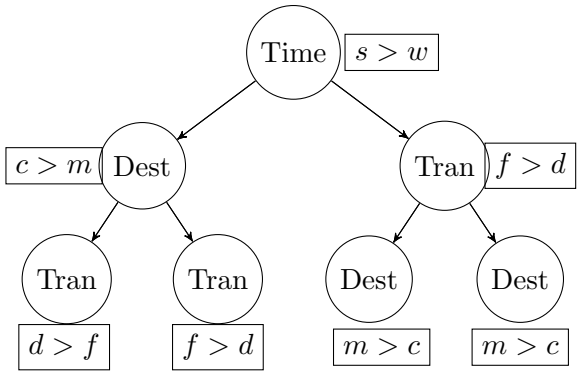


Figure 6.1: An LP tree T

decided to focus on three binary issues. The *Time* (X_1) of the travel could be either *summer* (s or 1_1) or *winter* (w or 0_1), the *Destination* (X_2) could be either *Chicago* (c or 1_2) or *Miami* (m or 0_2), and for *Transportation* (X_3) they could chooe to *drive* (d or 1_3) or *fly* (f or 0_3).

Jane, a member of the group, prefers a summer vacation to a winter one, and *Time* is the most important issue for her. If it is a summer trip, the next most important issue for Jane is *Destination* and she prefers Chicago to Miami. The least important issue is *Transportation*. She prefers driving if they go to Chicago, and flying, otherwise. For a winter trip, *Transportation* is more important than *Destination*. Jane does not like driving in the winter. As for the *Destination*, she prefers Miami to avoid the cold weather. All these preferences can be captured by the LP tree T in Figure 6.1(a). We note that the trees ordering the vacation plans for summer and for winter are determined by trees with different assignments of issues to nodes (the importance of nodes depends on the time of the trip). In particular, for the summer trips, the destination is the most important factor while for the winter trips the mode of transportation. The tree shows that the preferred vacation plan for Jane is to drive to Chicago in the summer and the next in order of preference is to fly to Chicago in the summer. The least preferred plan is to drive to Chicago in the winter.

A full representation of an LP tree requires as much space as an explicit enumeration of the preference order. However, sometimes LP trees can be represented in a much more concise way. For instance, if for some node t, its two subtrees are identical (that is, the corresponding nodes are assigned the same issue), they can be collapsed to a single subtree, with the same assignment of issues to nodes. This subtree is now a single child of its parent and it is neither left nor right, as it is associated with *each* of the two values of its parent node issue. To retain the preference information, at each node t' of the subtree we place a *conditional preference table*, and each preference in it specifies the preferred value for the issue labeling t' given the value of the issue labeling t. In the extreme case when for every node its two subtrees are identical, the tree can be collapsed to a path.

We now formally extend the definition of an LP tree to cover the case with collapsed subtrees. From now on, an *LP tree* is a tree in which every node except for the leaves has either two children, the left child and the right child, or a single "straight down" child, representing the collapse of the subtrees. The nodes are labeled with issues and, as before, we assume that each issue appears exactly once on each path from the root to a leaf.

Further, each node in an LP tree is assigned a *conditional preference table* (CPT) compensating for the loss in the structure due to applications of collapse higher in the tree. To describe CPTs, we need more terminology. For a node t, let $NonInst(t)$ be the set of ancestor nodes of t with just one child, and let $Inst(t)$ represent the remaining ancestor nodes of t. Being at a node t in an LP tree, we know the values of the issues on t's ancestor nodes with two children (in each such node we descend to the the left or to the right based on the value of that issue and the preference information at that node). Thus, the preferred value at t on issue $Iss(t)$ depends only on values of some of (possibly all) issues of the ancestor nodes of t that have only one child (belong to $NonInst(t)$). A *parent* function specifies these nodes. That is,

a parent function $\mathcal{P}$ assigns to each node t in T a set $\mathcal{P}(t) \subseteq \mathit{NonInst}(t)$ of *parents* of t, that is, the nodes in $\mathit{NonInst}(t)$ whose issues may have influence on the local preference at t. For each combination of values of the issues of nodes in $\mathcal{P}(t)$, the CPT for t contains a row specifying both that combinaton and the corresponding preferred value on the issue of t.

Since the preferred value for the issue at a node t may depend on the values of all issues above t, the conditional preference table for the node t located at distance i from the root may have as many as 2^i rows, (more exactly, 2^j rows, where j is the number of ancestor nodes with one child only). Thus, collapsing subtrees does not necessarily lead to smaller representation sizes. Collapsing may drastically reduce the size of the tree (even down to a simple path), but potentially at a cost of an exponential increase in the size of preference tables. (EXAMPLE)

Clearly, the conditional preference table at t requires $2^{|\mathcal{P}(t)|}$ rows. In the worst case, this number may be as high as 2^i, where i is the number of ancestors of t. But if each node has a bounded (independent of p) number of parents, all CPTs can be represented in $O(p)$ space. Thus, indeed, LP trees may offer a drastic reduction in the size of a representation of a vote over the domain $\mathcal{X}(\mathcal{I})$.

(THE LAST THREE PARAs should be illustrated with examples)

We can now introduce a useful classification of LP trees. If for every node t in an LP tree, $\mathcal{P}(t) = \emptyset$, all (local) preferences are unconditional and all CPTs consist of a single entry. Such trees are called *unconditional preference* LP trees (or UP trees, for short). Similarly, LP trees with all non-leaf nodes having exactly one child, are called an *unconditional importance* LP trees (UI trees, for short). Combining these characteristics leads to four classes of LP trees: unconditional importance and unconditional preference LP trees (UI-UP trees), unconditional importance and conditional preference trees (UI-CP trees), etc. The class of CI-CP trees comprises all LP trees, the class of UI-UP trees is the most narrow one.

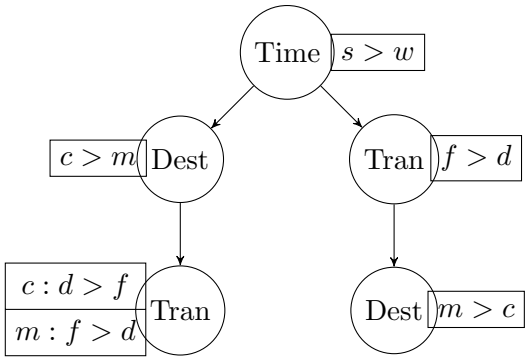


Figure 6.2: An CI-CP LP tree v

The LP tree T in Figure 6.1 can be represented more concisely as a (collapsed) CI-CP tree v in Figure 6.2. Nodes at depth one have their subtrees collapsed. In the tree in Figure 6.1, the subtrees of the node at depth 1 labeled *Tran* are not only identical but also have the same preference information at every node. Thus, collapsing them does not incur any growth in the size of the conditional preference table.

Finally, we recall the concept of a profile and of a positional scoring rule. A set of votes (collected from, say, n voters) over a domain $\mathcal{X}$ is called a *profile*. Among many rules proposed to aggregate a profile into a single preference ranking representing the group, positional scoring rules have received particular attention. For profiles over a domain with m alternatives, a *scoring vector* is a sequence $w = (w_0, \ldots, w_{m-1})$ of integers such that $w_0 \geq w_1 \geq \ldots \geq w_{m-1}$ and $w_0 > w_{m-1}$. Given a vote v with the alternative o in position i ($0 \leq i \leq m-1$), the score of o in v is given by $s_w(v, o) = w_i$. Given a profile $\mathcal{V}$ of votes and an alternative o, the score of o in $\mathcal{V}$ is given by

$$s_w(\mathcal{V}, o) = \sum_{v \in \mathcal{V}} s_w(v, o).$$

These scores determine the ranking generated from $\mathcal{V}$ by the scoring vector w (assuming, as is common, some independent tie breaking rule). In this paper we consider three positional scoring rules:

1. Borda: $(m-1, m-2, \ldots, 1, 0)$

2. k-approval: $(1, \ldots 1, 0, \ldots 0)$ with k the number of 1's

3. 2-valued (k, l)-approval: $(a, \ldots, a, b, \ldots, b, 0 \ldots, 0)$, where a and b are constants $(a > b)$ and the numbers of a's and b's equal to k and l, respectively.

6.3 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 an index i, $0 \leq i \leq m-1$, (1) returns the value w_i of the scoring vector, and (2) works in time polynomial in the sizes of i and m. Borda, k-approval and (k, l)-approval are examples of effective implicit positional scoring rules.

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 issues. 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 [57]. However, if k equals 2^{p-2} or 2^{p-3}, the problem is NP-complete, again for all four types of profiles [57] (in fact, the result holds for a larger set of values k, we refer for details to the paper by Lang et al. [57]). 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 24. *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 issues $X_1, \ldots, X_p$, 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 25. *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 issues $X_1, \ldots, X_p$, 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 issue 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. □

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, $\mathcal{I} = \{X_1,\ldots,X_p\}$). If $b \geq 2^p - 1$, b-Borda can be reduced to the (standard) Borda rule. Therefore, we focus here on the case when $b < 2^p - 1$.

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 [57]. We show that for some values of $b < 2^p - 1$, 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 Borda rule are, as we noted, in P). We start with the latter.

Theorem 26. *Let $b = 2^{p-1} - 1$, where p is the number of issues. The evaluation and the winner problems under b-Borda for profiles consisting of UI-UP LP trees 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 MIN-2SAT problem. DEFINE IT HERE, CITE THE PAPER THAT PROVES NP-COMPLETENESS.

Given a MIN-2SAT 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_{p+1} and define the set of issues $\mathcal{I}$ by setting $\mathcal{I} = \{X_1, \ldots, X_p, X_{p+1}$.

Second, for each $C_i \in \Phi$, we now build a set P_i of 12 UI-UP LP-trees over $\mathcal{I}$ (DO NOT USE AN EXAMPLE; GIVE A GENERAL DEFINITION) As an example, let C_i be $\neg X_2 \vee X_4$.

The fragment of the profile determined by C_i is given by the sequence

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

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 6.5. (REDO IT

INTO A THREE ROW FIGURE). 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 concatenation of all profiles P_i, $1 \leq i \leq m$. That is, $P = P_1 \circ P_2 \circ \ldots \circ P_m$. 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 $\mathcal{I}$. (HALF-BORDA NOT INTRODUCED, NOTATION s_{HB} NOT INTRODUCED). We have $s_{HB}(P_i, o) = 15a$, if $o \models X_{p+1} \wedge \neg C_i$; $s_{HB}(P_i, o) = 3a$, if $o \models X_{p+1} \wedge C_i$; $s_{HB}(P_i, o) < 15a$, if $o \models \neg X_{p+1} \wedge \neg C_i$; and $s_{HB}(P_i, o) < 3a$, if $o \models \neg X_{p+1} \wedge C_i$. (EXPLAIN WHY; OTHERWISE, YOU ARE NOT MAKING ANY USE OF THE TREE NOTATION ABOVE)

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 $o = (v, 1_{p+1})$. 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 $o \models \neg X_{p+1}$, we could flip the value on X_{p+1} from 0_{p+1} to 1_{p+1}, 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 $o \models X_{p+1}$, we are done. (YOU ARE USING SYMOBOLS $\models$ AND CONCEPTS SUCH AS SATISFY, BUT THEY NEED TO BE EXPLAINED.) $\square$

Corollary 27. *Theorem 26 holds for b-Borda when $b = 2^{p-c} - 1$, where c is a constant and $1 \leq c < p$.*

WHY - YOU NEED TO GIVE AN ARGUENT HERE

Theorem 28. *Let $b = 2^{p-c} - 1$, where p is the number of issues and c a fixed integer such that $1 \leq c < p$. The evaluation and the winner problems under b-Borda for*

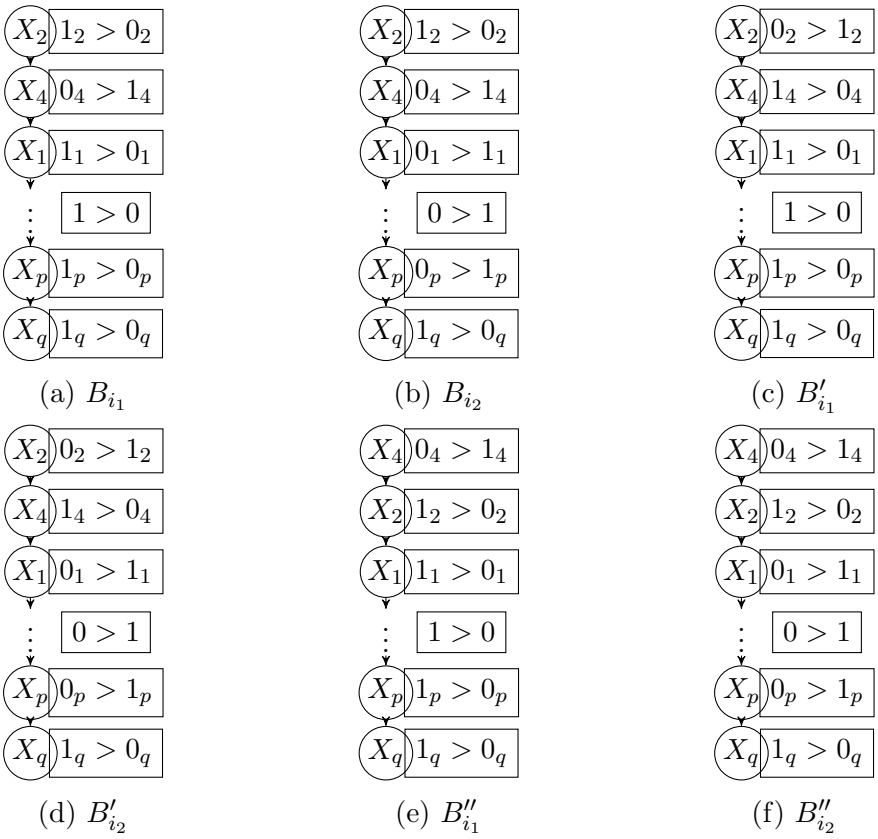


Figure 6.3: UI-UP LP trees

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 [57].

Given an instance $\langle I, P, l\rangle$ of $Borda^{ev}_{CI-UP}$, where I is a set of p issues $X_1, \ldots, X_p$, $P = \langle T_1, \ldots, T_m$ 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 issues. 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 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. □

(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 [60]. 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 29. *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] [54], which is NP-complete. Given an instance $\langle \Phi, l \rangle$ of the 2-MINSAT problem, we construct the set of issues $\mathcal{I}$, 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 issues 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 issue 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 6.4a. Secondly, we construct b duplicate trees shown in Figure 6.4b. Thirdly, we build another b duplicate trees shown in Figure 6.4c. (In all three figures we only indicate the top two issues since the other issues 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 $\mathcal{V}$.

[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.

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 $l' \geq n - l$. □

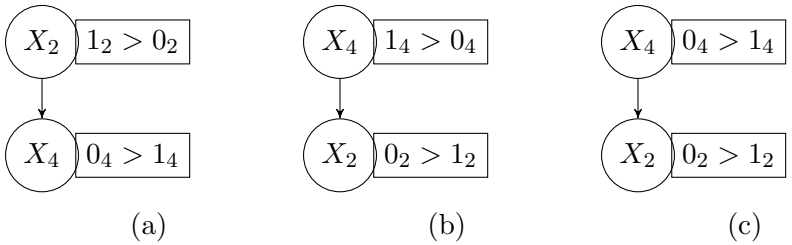


Figure 6.4: UI-UP LP trees

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. In general, the evaluation problem according to $(2^{p-c}, 2^{p-c})$-approval, where c is a constant and $1 < c < p$, for the four classes of LP trees is NP-complete. In each case, the hardness follows from from an NP-complete version of the *c-MINSAT* problem. Below we show the proof of NP-completeness of the evaluation problem for $(2^{p-3}, 2^{p-3})$-approval.

Theorem 30. *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* [70]. 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 issues 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 6.5a, a^2 duplicate trees of type 6.5b, a^2 duplicate trees of type 6.5c, a^2 duplicate trees of type 6.5d, $a^2 - ab$ duplicate trees of type 6.5e, $a^2 - ab$ duplicate trees of type 6.5f and $(a-b)^2$ duplicate trees of type 6.5g. 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 $l' \geq l$.

□

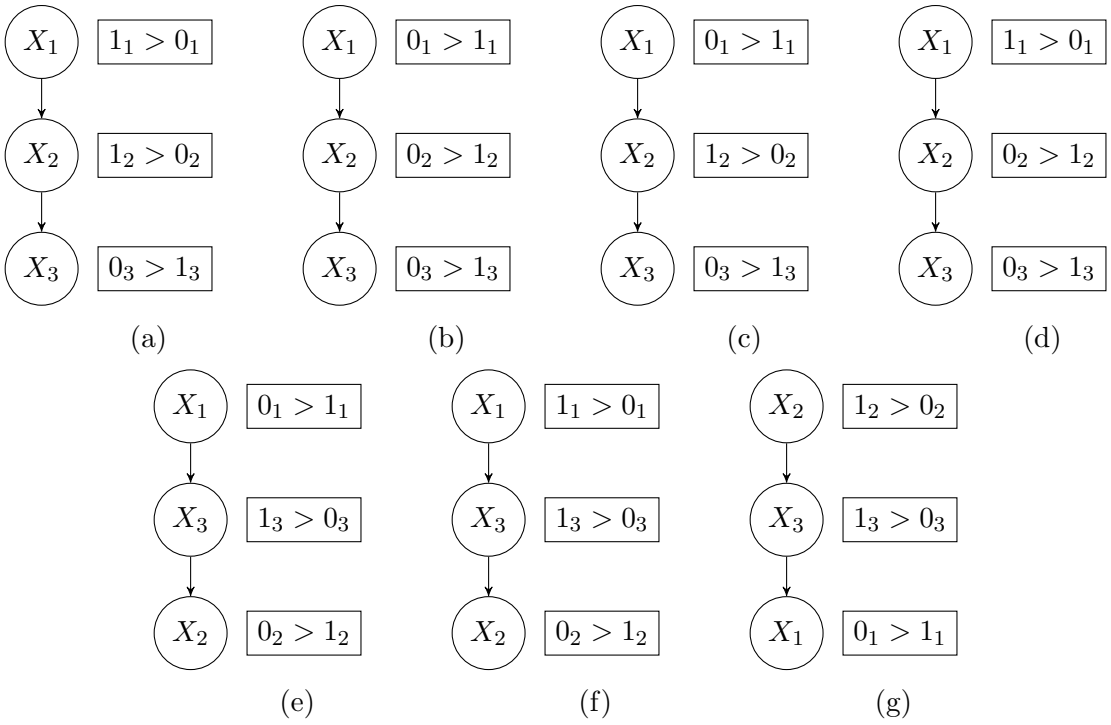


Figure 6.5: UI-UP LP trees

	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 $k \neq 2^{p-1}$

Figure 6.6: k-Approval

	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 $1 \leq c < p$

Figure 6.7: b-Borda

Computational Complexity Results

Now we summerize known computational complexity results of the evaluation problem for k-approval Figure 6.6, b-Borda Figure 6.7 and (k, l)-approval Figure 6.8 rules. The results for 2^{p-1}-approval, as well as those in Figure 6.6b and Figure 6.7a are attributed to Lang et al. [57].

	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 $1 < c < p$

Figure 6.8: (k, l)-Approval

6.4 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* [47], and *clingcon* [68] 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 issue X_i.

If X_i is the issue 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 issues 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

$$\begin{aligned} vote(v, d_i^v, i, 1) \text{ :- } & eval(i_1, x_{i_1}), \ldots, eval(i_j, x_{i_j}), \\ & eval(i_{j+1}, x_{i_{j+1}}), \ldots, eval(i_\ell, x_{i_\ell}) \end{aligned} \tag{6.1}$$

(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 issue 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 6.2 is translated into the logic program in Figure 6.9 (*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 6.9: 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 6.10). Parameters in the evaluation problem are defined as facts (lines 1-4):

```
1  issue(1). issue(2). issue(3).
2  numIss(3).
3  val(0). val(1).
4  threshold(5).
5  1{ eval(I,M) : val(M) }1 :- issue(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 6.10: Borda evaluation problem encoding in *clingo*

predicates *issue/1*s representing three issues, *numIss/1* the number of issues, *threshold/1* the threshold value, together with *val/1*s the two values in the issues' binary domains. Line 5 generates the search space of all alternatives over three binary issues. It expresses that if X is an issue, 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 issue 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

$$s_B(v, o) = \sum_{i=1}^{p} 2^{p-d_i^v} \cdot f(a_i, x_i), \tag{6.2}$$

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

$$s_B(\mathcal{V}, o) = \sum_{v=1}^{n} \sum_{i=1}^{p} 2^{p-d_i^v} \cdot f(a_i, x_i). \tag{6.3}$$

In the program in Figure 6.10, 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 (6.3), if issue 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 issues; if issue 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 issue 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 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 6.10 with the following single rule:

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

```
1  $domain(1..4).
2  issue(1). issue(2). issue(3).
3  numIss(3).
4  val(0). val(1).
5  threshold(5).
6  1{ eval(I,M) : val(M) }1 :- issue(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 6.11: Borda evaluation problem encoding using *clingcon*

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* [74]). In Figure 6.11 we encode the Borda evaluation problem in *clingcon*.

Lines 2-8 are same as lines 1-7 in Figure 6.10. 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 issues in an LP tree of 3 issues are 2^0, 2^1 and 2^2.

The encoding for the Borda winner problem for *clingcon* replaces rules 10 in Figure 6.11 with the following one rule:

```
$maximize{weight(V,I):voteID(V):issue(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 31.

Theorem 31. *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 issues 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 issue $X_{i_j} \in IO_{\vec{d}_k}$ 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 issue I at depth D. For the example LP tree in Figure 6.2, we have the follow ancillary logic program in Figure 6.12.

```
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 6.12: Auxiliary data in logic rules for computing $\vec{d_k}$

We now present the encoding of the k-Apprvoal evaluation problem in *clingo* (Figure 6.13), where $k = 5$.

6.5 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

```
1  issue(1). issue(2). issue(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 :- issue(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 6.13: k-Approval evaluation problem encoding in *clingo*

Satisfiability instances, which are then transformed into Weighted Partial Maximum Satisfiability instances.

Weighted Partial Maximum Satisfiability

Definition 48. 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 49. Let X be a set of boolean variables $\{X_1, \ldots, X_p\}$, a *weighted partial formula* Φ [3] is a multiset of weighted clauses over X of the form

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

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

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 50. 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 [45], the MAXSAT problem [27] and the Partial MAXSAT problem [27].

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 32. *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.

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 6.2 under Borda is tranlated to a WTM instance in Figure 6.14.

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

Figure 6.14: The WTM instance of the LP tree v

Then the WTM instance in Figure 6.14 is transformed into a WPM instance in Figure 6.15.

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

Figure 6.15: The WPM instance of the LP tree v

Encoding k-approval problems in WTM and WPM

The LP tree in Figure 6.2 under 5-Approval is translated to a WTM instance in Figure 6.16.

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

$(X_1, 1)$

Figure 6.16: The WTM instance of the LP tree v under 5-Approval

Then the WTM instance in Figure 6.16 is transformed into a WPM instance in Figure 6.17.

$(C_1, 1)$

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

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

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

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

$(C_2, 1)$

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

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

Figure 6.17: The WPM instance of the LP tree v under 5-Approval

6.6 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 issues (5/10/20) and for each number of issues 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 issues 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 $< d_s$.[4]

The algorithm starts by randomly selecting issues 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 issues) 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 6.18 depicts a CI-CP tree of 4 issues in this class.

Solving the Winner Problem

We refer to 6.19 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 issues are consistent with the property that the problem is solvable in polynomial time. All three solvers scale up well. Figures 6.19(a),(c) and

[4]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 issues.

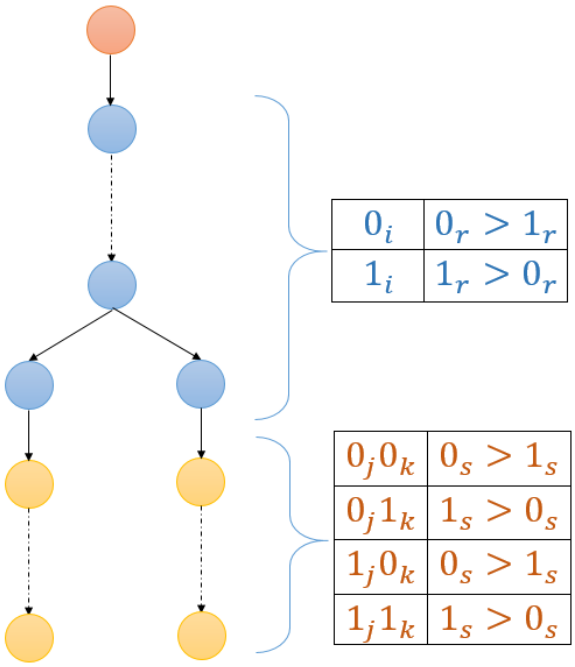


Figure 6.18: *Simple* CI-CP tree

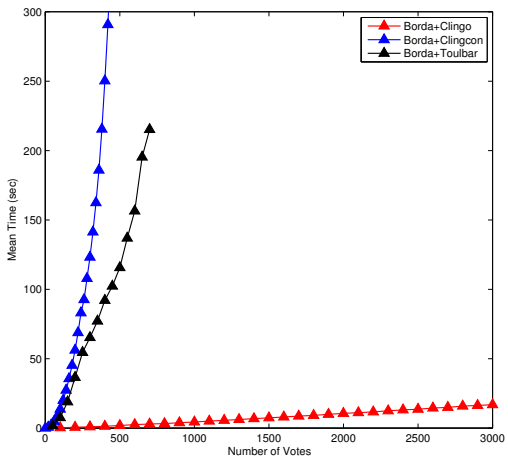
(e) depict the result for the cases with 10 issues. When we fix the number of votes and vary the number of issues the time grows exponentially with p (cf. Figures 6.19(b),(d) and (f)), again consistently with the computational complexity of the problems (NP-hardness).

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.

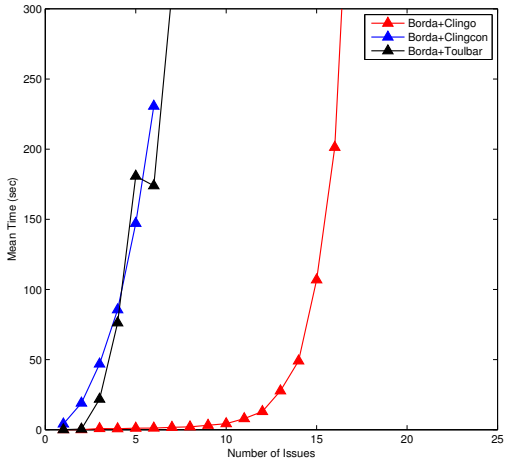
For the winner problems, our experiments demonstrates that profiles of LP trees of practical sizes can be effectively handled by our solvers, up to 3000 votes per profile over up to 20 issues. But going beyond 20 issues remains a challenge.

Solving the Evaluation Problem

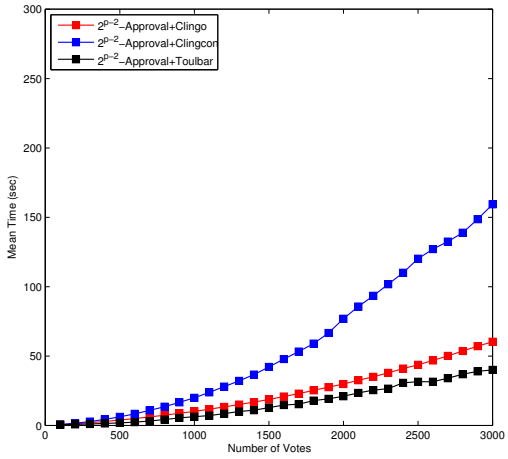
The evaluation problem can be reduced to the winner problem, as an evaluation problem instance has an answer *YES* if and only if the score of the winner equals or



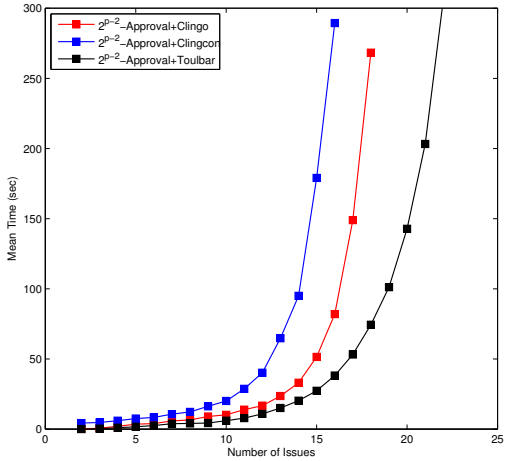
(a) Borda, fixed #issues(10)



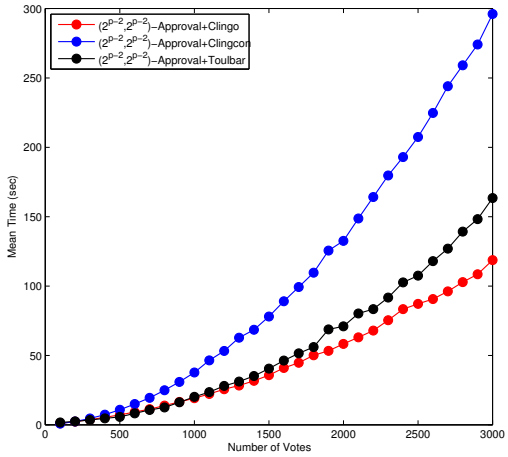
(b) Borda, fixed #votes(1000)



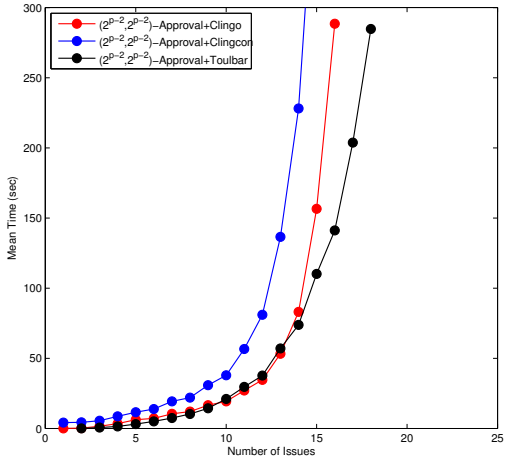
(c) 2^{p-2}-Approval, fixed #issues(10)



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



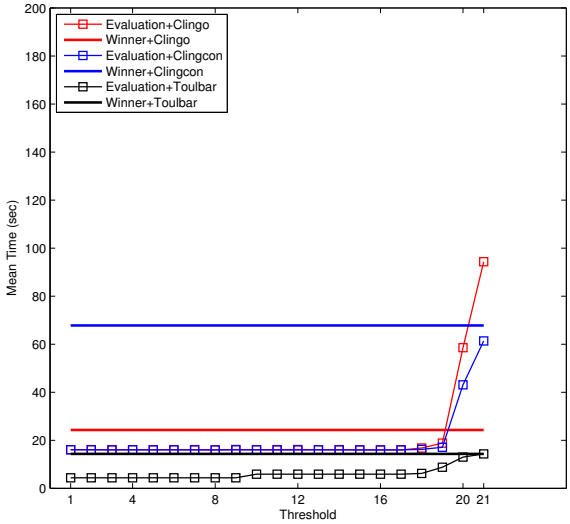
(e) $2K$-Approval, fixed #issues(10)



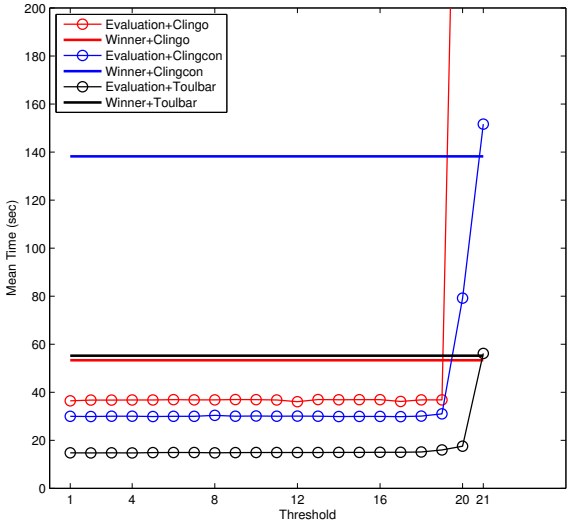
(f) $2K$-Approval, fixed #votes(1000)

Figure 6.19: Solving the winner problem given *simple* LP trees

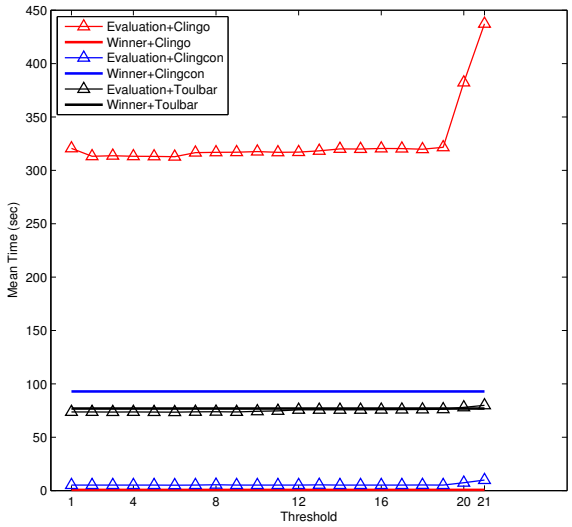
exceeds the threshold. Thus, the evaluation problem is at most as complex as the winner problem.



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



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



(c) Borda (1000votes/4issues)

Figure 6.20: Solving the evaluation problem given *simple* LP trees

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 6.20(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 winning score and the evaluation problem becomes harder when the thresholds are close to it. We refer to Figures 6.20(a), (b) and (c).

Thirdly, in all cases *clingcon* outperforms *clingo* on the evaluation problems (cf. Figures 6.20(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 6.20(a), (b) and (c).

6.7 Conclusions and Future Work

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. [57]; 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 issues is fixed the ASP tools scale up consistently with the polynomial time complexity. In general, the tools are practical even if the number of issues is up to 15 and the number of votes is as high as 500. This is remarkable as 15 binary issues 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 [19] and answer set optimization (ASO) preferences [25].

Chapter 7 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[52]

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[79], 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, 22]:
   - Voting
   - Fair division
   - Strategyproof Social Choice
2. Automated Planning and Scheduling[76, 17, 15]:
   - 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] John J Bartholdi III and James B Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

[15] 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.

[16] Nadja Betzler, Rolf Niedermeier, and Gerhard J Woeginger. Unweighted coalitional manipulation under the borda rule is np-hard. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume One*, pages 55–60. AAAI Press, 2011.

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

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

[19] 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.

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

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

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

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

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

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

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

[27] 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.

[28] Jessica Davies, George Katsirelos, Nina Narodytska, and Toby Walsh. Complexity of and algorithms for borda manipulation. In *AAAI*, volume 11, pages 657–662, 2011.

[29] 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.

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

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

[32] Carmel Domshlak, Steven David Prestwich, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *J. Heuristics*, 12(4-5):263–285, 2006.

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

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

[35] Yagil Engel and Michael P. Wellman. Cui networks: A graphical representation for conditional utility independence. In *AAAI*, pages 1137–1142. AAAI Press, 2006.

[36] Wolfgang Faber, Miroslaw Truszczynski, and Stefan Woltran. Strong equivalence of qualitative optimization problems. In *KR*, 2012.

[37] Wolfgang Faber, Miroslaw Truszczynski, and Stefan Woltran. Abstract preference frameworks - a unifying perspective on separability and strong equivalence. In *AAAI*, 2013.

[38] Piotr Faliszewski, Edith Hemaspaandra, and Henning Schnoor. Manipulation of copeland elections. In *In AAMAS-10*, pages 367–374, 2010.

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

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

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

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

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

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

[45] 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.

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

[47] 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.

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

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

[50] 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.

[51] 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.

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

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

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

[55] Mark W. Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.*, 36(3):490–509, 1988.

[56] Jérôme Lang, Jérôme Mengin, and Lirong Xia. Aggregating conditionally lexicographic preferences on multi-issue domains. In *CP*, pages 973–987, 2012.

[57] Jérôme Lang, Jérôme Mengin, and Lirong Xia. Aggregating conditionally lexicographic preferences on multi-issue domains. In *CP*, pages 973–987, 2012.

[58] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 1997.

[59] Xudong Liu and Miroslaw Truszczynski. Aggregating conditionally lexicographic preferences using answer set programming solvers. In *ADT*, pages 244–258, 2013.

[60] 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.

[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] E. Muller and M. Satterthwaite. he equivalence of strong positive association and strategy-proofness. *Journal of Economic Theory*, 14(2):412–418, 1977.

[66] Pierfrancesco La Mura and Yoav Shoham. Expected utility networks. *CoRR*, abs/1301.6714, 2013.

[67] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25:241–273, 1999.

[68] Max Ostrowski and Torsten Schaub. ASP modulo CSP: The clingcon system. *TPLP*, 12(4-5):485–503, 2012.

[69] C. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Inc., 1994.

[70] 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.

[71] Z. Ruttkay. Fuzzy constraint satisfaction. In *Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on*, pages 1263 –1268 vol.2, jun 1994.

[72] 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.

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

[74] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Modeling and programming with gecode, 2010.

[75] Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *TPLP*, 6(5):559–607, 2006.

[76] Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 2006.

[77] Yves Sprumont. The division problem with single-peaked preferences: a characterization of the uniform allocation rule. *Econometrica*, 59(2):509–519, 1991.

[78] Giorgio Gosti Stefano Bisterelli and Francesco Santini. Solving fuzzy dcsps with naming games.

[79] Ali Fallah Tehrani, Weiwei Cheng, and Eyke Hüllermeier. Choquistic regression: Generalizing logistic regression using the choquet integral. In *EUSFLAT*, 2011.

[80] Wikipedia. Social choice theory, 2013.

[81] Nic Wilson. Consistency and constrained optimisation for conditional preferences. In *ECAI*, pages 888–894, 2004.

[82] Nic Wilson. Extending cp-nets with stronger conditional preference statements. In *in Proceedings of AAAI-04*, pages 735–741, 2004.

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

[84] Lirong Xia and Vincent Conitzer. Approximating common voting rules by sequential voting in multi-issue domains. In *ISAIM*, 2012.

[85] Lirong Xia, Michael Zuckerman, Ariel D Procaccia, Vincent Conitzer, and Jeffrey S Rosenschein. Complexity of unweighted coalitional manipulation under some common voting rules. In *IJCAI*, volume 9, pages 348–352, 2009.

[86] 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.

Vita

A brief vita goes here.