Aggregating Conditionally Lexicographic Preferences Using Answer Set Programming Solvers

Xudong Liu and Miroslaw Truszczynski

University of Kentucky
Lexington, KY 40506, USA
liu@cs.uky.edu, mirek@cs.engr.uky.edu

Abstract. We consider voting over combinatorial domains, where alternatives are binary tuples. We assume that votes are specified as *conditionally lexicographic preference trees*, or *LP trees* for short. We study the aggregation of LP tree votes for several positional scoring rules. Our main goal is to demonstrate that answer-set programming tools can be effective in solving the *winner* and the *evaluation* problems for instances of practical sizes. To this end, we propose encodings of the two problems as answer-set programs, design methods to generate LP tree votes randomly to support experiments, and present experimental results obtained with ASP solvers *clingo* and *clingcon*.

Keywords: conditionally lexicographic preferences, social choice theory, positional scoring voting rules, answer set programming.

1 Introduction

Preferences are an essential component in areas including constraint satisfaction, decision making, and social choice theory. Modeling and reasoning about preferences are crucial to these areas. Consequently, several preference formalisms have been developed, such as penalty logic, possibilistic logic, and conditional preference networks (*CP nets*, for short) [6]. Each of these formalisms provides the user with a concise way to express her preferences.

The problem of aggregating preferences of a group of users (often referred to as *voters*) is central to decision making and has been studied extensively in social choice theory. While in the cases when the number of alternatives is small the problems of computing the winner and of deciding whether there is an outcome with the score higher than a given threshold are polynomially solvable for the majority of commonly considered voting rules, the situation changes when we consider votes over combinatorial domains of binary p-tuples. Since the number of outcomes grows exponentially with p, applying voting rules directly is often infeasible. Issue-by-issue voting approximates voting rules by considering only the first choices in votes, but experiments show that the winners selected issue by issue are often not the winners selected by common voting rules [4].

When votes are represented as *conditionally lexicographic preference trees*, or LP trees, for short [1], the problem of computing the winning alternative, or

P. Perny, M. Pirlot, and A. Tsoukiàs (Eds.): ADT 2013, LNAI 8176, pp. 244–258, 2013.

the *winner* problem, is generally NP-hard, while a related problem to decide whether there is an alternative with the score exceeding a given threshold, the so called *evaluation* problem, is NP-complete [8]. Nevertheless, the two problems arise in practice and computational tools to address them effectively are needed. In this work we study Borda, k-approval and 2-valued (k, l)-approval scoring rules. For the 2-valued (k, l)-approval, we obtain new complexity results. For all three rules we encode the problems in answer-set programming [9] and study the effectiveness of ASP solvers *clingo* [5] and *clingcon* [10] in solving the winner and the evaluation problems. We chose these two solvers as they represent substantial different approaches to computing answer sets. The former, *clingo*, is a native ASP solver developed along the lines of satisfiability solvers. The latter, *clingcon*, enhances *clingo* with specialized treatment of some common classes of numeric constraints by delegating some reasoning tasks to a CP solver *Gecode* [12]. As problems we are considering involve numeric constraints, a comparison of the two solvers is of interest. To support the experimentation we propose and implement a method to generate LP votes, of some restricted form, randomly.

The main contributions of our work are: (1) new complexity results for the winner and the evaluation problems for a class of positional scoring rules; and (2) demonstration that ASP is an effective formalism for modeling and solving problems related to aggregation of preferences given as LP trees.

2 Technical Preliminaries

A *vote* over a set of *alternatives* (or *outcomes*) $\mathcal{X}$ is a *strict total* order $\succ$ on $\mathcal{X}$. Here we consider votes 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. Thus, even for moderately small values of p, eliciting precise orders over all alternatives and representing them directly may be infeasible. Instead, in cases when preferences have some structure, that structure can be exploited to give rise to concise preference expressions [2,8].

In this work we study the case when votes (preferences) are given as LP trees [1]. 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). 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 standing for 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 tree T in Figure 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

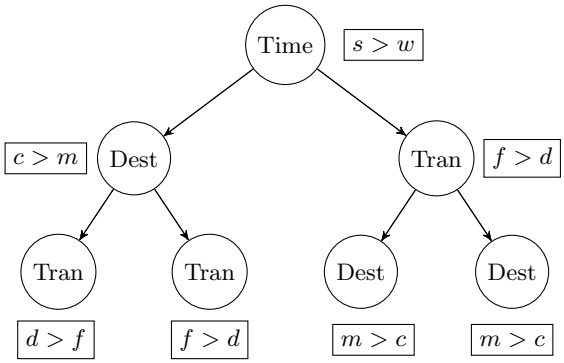


Fig. 1. An LP tree T

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 1 can be represented more concisely as a (collapsed) CI-CP tree v in Figure 2. Nodes at depth one have their subtrees collapsed. In the tree in Figure 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 growth in the size of the conditional preference table.

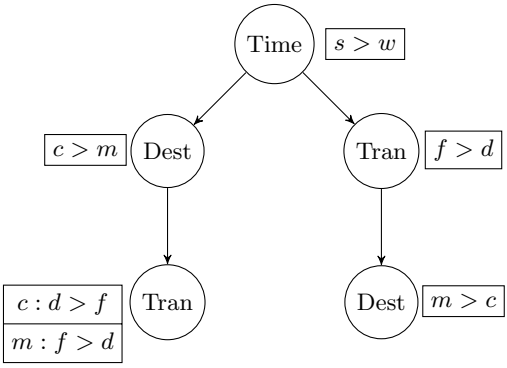


Fig. 2. An CI-CP LP tree v

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.

3 The Problems and Their Complexity

Here we consider only *effective implicit* positional scoring rules, that is, rules defined by an algorithm that given m (the number of alternatives) and i, $0 \leq i \leq$

$m-1$ (an index into the scoring vector) returns the value w_i of the scoring vector and 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.

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 [8]. The picture for the the k-approval rule is more complicated. If $k = 2^{p-1}$ the evaluation problem is in P for all four classes of profiles. However, if k equals 2^{p-2} or 2^{p-3}, the problem is NP-complete, again for all four LP profile types [8] (in fact, the result holds for a larger set of values k, we refer for details to Lang et al. [8]). Clearly, in each case where the evaluation problem is NP-complete, the winner problem is NP-hard.

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. We describe two such cases here: (1) $k = l = 2^{p-2}$, and (2) $k = l = 2^{p-3}$. We note that if $a = 1$ and $b = 0$, case (1) reduces to 2^{p-2}-approval and case (2) to 2^{p-3}-approval. If $a = 2$ and $b = 1$, we refer to the rule in case (1) as $2K$-*approval*.

Theorem 1. *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*[1] [7], 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.

[1] 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.

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 3a. Secondly, we construct b duplicate trees shown in Figure 3b. Thirdly, we build another b duplicate trees shown in Figure 3c. (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 $\mathcal{V}$.

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 $l' \geq n - l$. □

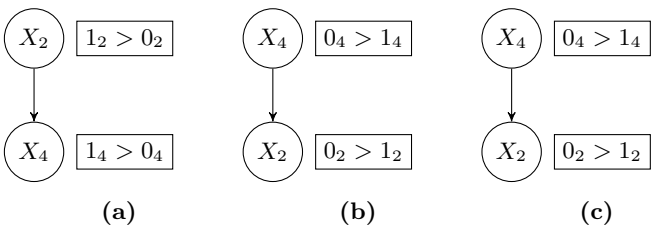


Fig. 3. UI-UP LP trees

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. The evaluation problem according to $(2^{p-3}, 2^{p-3})$-approval for the four classes of LP trees is also NP-complete. In this case, the hardness is proven by a reduction from an NP-complete version of the 3-*MINSAT* problem [11].

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) [9]. 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* [5], and *clingcon* [10] and demonstrate the effectiveness of ASP in modeling problems related to preference aggregation. We selected

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

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

(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 2 is translated into the logic program in Figure 4 (*voteID(v)* identifies the id of the vote (LP tree)).

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

Fig. 4. Translation of v 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 5). 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.
```

Fig. 5. 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

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

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

$$s_B(\mathcal{V}, o) = \sum_{v=1}^{n} \sum_{i=1}^{p} 2^{p-d_i^v} \cdot f(a_i, x_i). \tag{3}$$

In the program in Figure 5, lines 6 and 7 introduce predicate *wform/3* which computes $2^{p-d_i^v} \cdot f(a_i, x_i)$ used to compute Borda score. According to equation (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 Figure5 with the following single rule:

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

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

Encoding the Borda Evaluation Problem in *Clingcon*. In this encoding, we exploit *clingcon*'s ability to handle some numeric constraints by specialized constraint solving techniques (by means of the CP solver *Gecode* [12]). In Figure 6 we encode the Borda evaluation problem in *clingcon*.

```
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).
```

Fig. 6. Borda evaluation problem encoding using *clingcon*

Lines 2-8 are same as lines 1-7 in Figure 5. Line 9 defines the constraint variable *weight(V,I)* that assigns weight W to each pair (V,I) and line 10 defines a global constraint by use of *$sum* declares that the Borda score must be at least the threshold. Line 1 restricts the domain of all constraint variables (only *weight/2* in this case) to [1,4] as weights of 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 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)*.

5 Experiments and Results

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 $< d_s$.[2]

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 7 depicts a CI-CP tree of 4 issues in this class.

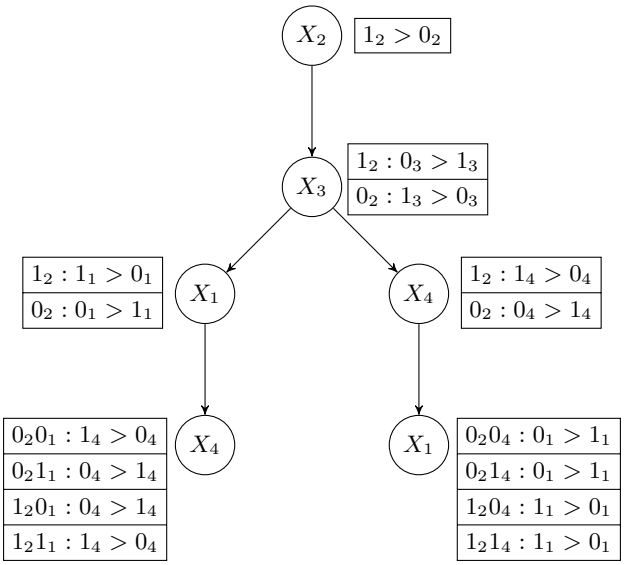
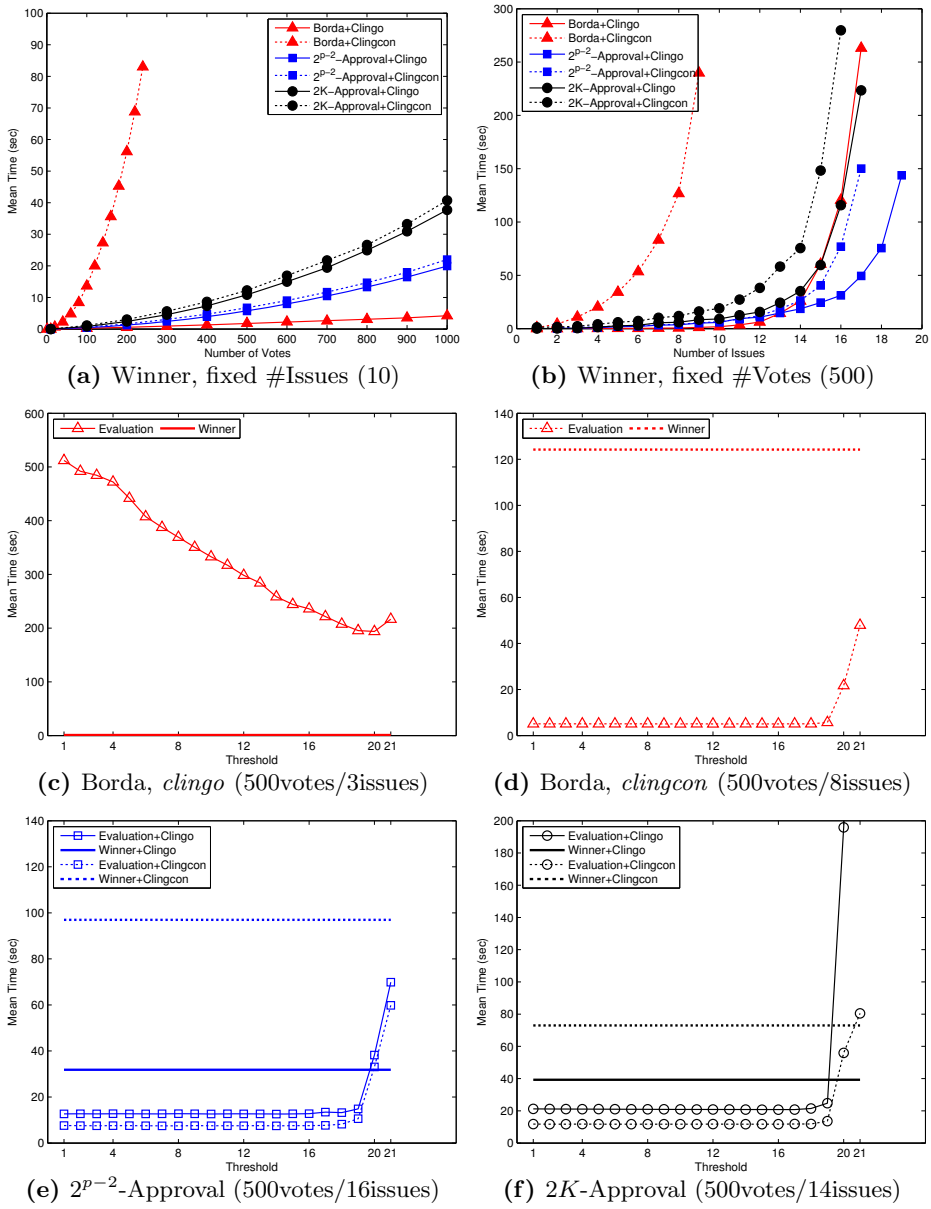


Fig. 7. A CI-CP tree of 4 issues

The goals of experimentation are to demonstrate the effectiveness of ASP tools in aggregating preferences expressed as LP trees, and to compare the performance of *clingo* and *clingcon*. We focus on three voting systems, Borda, 2^{p-2}-approval and $2K$-approval, for both the winner and the evaluation problems.

[2] 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.



(a) Winner, fixed #Issues (10)

(b) Winner, fixed #Votes (500)

(c) Borda, *clingo* (500votes/3issues)

(d) Borda, *clingcon* (500votes/8issues)

(e) 2^{p-2}-Approval (500votes/16issues)

(f) $2K$-Approval (500votes/14issues)

Fig. 8. Aggregating *simple* LP trees

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. Each test case (using *clingo 3.0.5* or *clingcon 2.0.3*) was performed with a limit of 10 minutes.

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 1000 for $\{Borda, 2^{p-2}\text{-}approval, 2K\text{-}approval\} \times \{clingcon, clingo\}$. Then we fix the number of votes (500) 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 10 tests over different randomly generated profiles of *simple* LP trees.

For the evaluation problem, we compare its experimental complexity with that of the winner problem. For each of the 10 randomly generated profiles, 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 the maximization construct in *clingo* (the main difference between the *winner* and the *evaluation* problems is in the use of that construct in the encoding of the former). We again present and compare average time results.

Varying the Number of Issues and the Number of Votes. Our experiments on the winner problem for the three voting rules with the fixed number of issues are consistent with the property that the problem is solvable in polynomial time. Both *clingo* and *clingcon* scale up well. Figure 8a depicts the results 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. Figure 8b), 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. We attribute that first to the use of the optimization construct in *clingo*, which allows us to keep the size of the ground propositional theory low, and second to the effective way in which optimization constructs are implemented in that system. Thus, in these examples, the main benefit of *clingcon*, its ability to avoid grounding and preprocessing by "farming out" some of the solving job to a dedicated constraint solver, does not offer *clingcon* the edge. Finally, for both *clingo* and *clingcon*, Borda is the hardest rule to deal with, especially when the number of issues is large.

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

We compared the two problems by first solving the winner problem, and then solving the evaluation problem on the same instances with the value of the threshold growing at the step of 5% of the winner score. That gives us 20 normalized points for each instance. In the last run (point 21 on the x-axis) we used the winner's score plus 1 as the threshold to determine the optimality of the winner's score. These experiments allow us to compare the hardness of the winner problem (more precisely, the effectiveness of solvers) with that of the evaluation problem.

First, we note that for *clingo*, the evaluation problem is harder than the winner problem in the entire range for Borda (Figure 8c), and for the two other rules, when the threshold is close to the winner's score or exceeds it (Figures 8e and 8f). 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.

For *clingcon* the situation is different. Figures 8d, 8e and 8f show that 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. It seems to suggest that the constraint solver used by *clingcon* performs well in comparison with the implementation of the optimization constructs in *clingcon*. Finally, in all cases *clingcon* outperforms *clingo* on the evaluation problems. 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.

6 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. [8]; 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 [2] and answer set optimization (ASO) preferences [3].

Acknowledgments. This work was supported by the NSF grant IIS-0913459.

References

1. Booth, R., Chevaleyre, Y., Lang, J., Mengin, J., Sombattheera, C.: Learning conditionally lexicographic preference relations. In: ECAI, pp. 269–274 (2010)
2. Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., Poole, D.: CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. Journal of Artificial Intelligence Research 21, 135–191 (2004)
3. Brewka, G., Niemelä, I., Truszczynski, M.: Answer set optimization. In: IJCAI, pp. 867–872 (2003)
4. Fargier, H., Conitzer, V., Lang, J., Mengin, J., Schmidt, N.: Issue-by-issue voting: an experimental evaluation. In: MPREF (2012)
5. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam answer set solving collection. AI Communications 24(2), 105–124 (2011)
6. Kaci, S.: Working with Preferences: Less Is More. In: Cognitive Technologies. Springer (2011)
7. Kohli, R., Krishnamurti, R., Mirchandani, P.: The minimum satisfiability problem. SIAM J. Discrete Math. 7(2), 275–283 (1994)
8. Lang, J., Mengin, J., Xia, L.: Aggregating conditionally lexicographic preferences on multi-issue domains. In: Milano, M. (ed.) CP 2012. LNCS, vol. 7514, pp. 973–987. Springer, Heidelberg (2012)
9. Marek, V.W., Truszczynski, M.: Stable models and an alternative logic programming paradigm. In: Apt, K.R., Marek, V.W., Truszczynski, M., Warren, D.S. (eds.) The Logic Programming Paradigm: a 25-Year Perspective, pp. 375–398. Springer, Berlin (1999)
10. Ostrowski, M., Schaub, T.: ASP modulo CSP: The clingcon system. TPLP 12(4-5), 485–503 (2012)
11. Papadimitriou, C., Yannakakis, M.: Optimization, approximation, and complexity classes. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC 1988, pp. 229–234. ACM, New York (1988)
12. Schulte, C., Tack, G., Lagerkvist, M.Z.: Modeling and programming with gecode (2010)