Aggregating Conditionally Lexicographic Preferences Using Answer Set Programming Solvers

Xudong Liu and **Miroslaw Truszczynski**
University of Kentucky, USA
liu@cs.uky.edu and 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*.

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) [Kaci, 2011]. 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 the social choice theory. While in the cases when the number of alternatives is small the problems of computing the winner and deciding whether there are outcomes with score equal to or exceeding a given threshold are polynomially solvable for the majority of commonly considered voting rules, the situation changes when we consider votes over combinatorial domains with p variables. 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 the experiments show that the winners selected issue by issue are often not the winners selected according to the voting rule [Fargier *et al.*, 2012].

When votes are represented as LP trees [Booth *et al.*, 2010], the problem of computing the winning alternative or, the *winner* problem, is generally NP-hard while a related problem to decide whether there is an alternative with the score equal to or exceeding a given threshold, the so called *evaluation* problem, is NP-complete [Lang *et al.*, 2012]. 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 latter kind, we obtain new complexity results. For all three rules we encode the problems in answer-set programming [Marek and Truszczynski, 1999] and study the effectiveness of ASP solvers *clingo* [Gebser *et al.*, 2011] and *clingcon* [Ostrowski and Schaub, 2012] in solving the winner and the evaluation problems. 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 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* over $\mathcal{I}$ 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 $\mathcal{I} = \{X_1, X_2, X_3\}$. Then, a 3-tuple $(0_1, 1_2, 1_3)$ is 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 $\mathcal{X}$, denoted by m, 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 the cases when preferences have some structure, that structure is exploited to give rise to concise preference encoding expressions [Boutilier *et al.*, 2004; Lang *et al.*, 2012].

In this work we study the case when votes are given as *LP trees* [Booth *et al.*, 2010]. An LP tree $\mathcal{L}$ over a set of p binary issues $\mathcal{I}$ consists of three components: (1) a *binary tree* T, where each node t is labeled by an issue $Iss(t) \in \mathcal{I}$ so that each issue appears exactly once on each path from the root to a leaf. For each node t in an LP tree T we denote

by $NonInst(t)$ (respectively, $Inst(t)$) the set of nodes that are ancestors of t in T that have one child (respectively, two children); (2) a *parent* function $\mathcal{P}$ that assigns to each node t in T a set of "parents" of t, $\mathcal{P}(t) \subseteq NonInst(t)$, that are nodes whose values determine local preferences associated with node t; and (3) for each node t, a *conditional preference table* $CPT(t)$ that specifies local preferences for $Iss(t)$, say X_i, by expressions $u : 1_i > 0_i$ or $u : 0_i > 1_i$, where u is a combination of values of issues labeling $\mathcal{P}(t)$.

Given an alternative $x_1x_2 \dots x_p$, we find its ranking in an LP tree T by traversing the tree from the root to a leaf. If we are at node t labeled with the issue X_i, we identify the row in the table $CPT(t)$ that corresponds to the values of all the issues that label the nodes in the set $\mathcal{P}(t)$. If x_i is the preferred value for X_i according to that row in the table, we descend to the left child of t; otherwise, the right child. The location of the leaf we end up in determines the ranking of the alternative, with the leftmost leaf being the highest rank.

If t has only one child, it indicates that the left and the right subtrees, including the *CPT*s of all nodes, are identical. Collapsing them to a single subtree is used only to keep the size of the tree as low as possible. Conceptually, the tree still has two children. It is clear that if we are in a node t in an LP tree, we know the values of all issues that label its ancestors with two children. Thus, if all ancestors of t have two children, then $CPT(t)$ has a single row specifying the preferred value for $Iss(t)$. Otherwise, some ancestors have a single child that may have any of the two values its issue can take. To decide whether to go left or right at t may depend on values of issues labeling some of those nodes. The parent function $\mathcal{P}$ identifies them and so, the $CPT(t)$ requires $2^{|\mathcal{P}(t)|}$ rows to cover all the possibilities.

Issues higher on a path have more importance. In particular, the value of the issue in the root determines whether an alternative ends in the top half or the bottom half of the ranking. If an LP tree consists of a path from the root down to a single leaf, then the importance of issues on every path that tree represents is the same. In such case, we speak about *unconditional importance*. Otherwise, importance is conditional. Similarly, if for every node t, $CPT(t)$ consist of a single row (the preference on the value of $Iss(t)$ does not depend on the values of the issues labeling its ancestors), we speak about *unconditional preferences*. Otherwise, preferences are conditional. This classification gives rise to four types of LP trees: unconditional importance and unconditional preferences (UI-UP), unconditional importance and conditional preferences (UI-CP), etc. The class CI-CP (conditional importance and conditional preferences) is most general and contains the other three classes. In general, the size of a CI-CP tree can still be exponential in the number of issues p. However, if there is a fixed (independent of p) bound on the number of nodes with two children and, similarly, there is a fixed bound on the number of parents a node can have, the size of the tree is linear in p. On the other hand, the UI-UP LP trees have always the size linear in p.

To illustrate these notions, let us consider a simple example. A group of people attempt to decide on their vacation plan for 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).

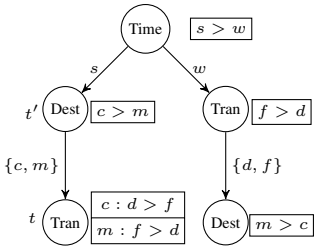


Figure 1: An LP tree $\mathcal{L}_1$ in the class CI-CP

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 knows she 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 CI-CP LP tree in Figure 1.

Let t be the bottom left node, t' the middle left node. We have $Iss(t)$ = *Transportation*, $Iss(t')$ = *Destination* and $NonInst(t) = \mathcal{P}(t) = \{t'\}$. The linear order represented by Jane's LP tree is $scd \succ scf \succ smf \succ smd \succ wfm \succ wfc \succ wdm \succ wdc$. Thus, the optimal alternative for Jane is to drive to Chicago during the summer break.

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 $w = (w_0, \dots, w_{m-1})$ of integers such that $w_0 \geq w_1 \geq \dots \geq w_{m-1}$ and $w_0 > w_{m-1}$. Given a vote $\mathcal{L}_v$ with the alternative o in position j, the score of o in $\mathcal{L}_v$ is given by $s_w(\mathcal{L}_v, o) = w_j$. Given a profile $\mathcal{V}$ of votes and an alternative o, the score of o in $\mathcal{V}$ is given by $s_w(\mathcal{V}, o) = \sum_{\mathcal{L}_v \in \mathcal{V}} s_w(\mathcal{L}_v, o)$. 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: $(m-1, m-2, \dots, 1, 0)$
2) k-approval: $(1, \dots 1, 0, \dots 0)$ with k the number of 1's
3) 2-valued (k, l)-approval: $(a, \dots, a, b, \dots, b, 0 \dots, 0)$, with $a > b$ and the numbers of a's and b's equal to k and l, respectively.

3 The Problems and Their Complexity

Let $\mathcal{V}$ be an LP profile, R a positive integer (a *threshold*), and w a scoring vector. The *winner* problem consists of computing an alternative $o \in \mathcal{X}$ with the maximum score $s_w(\mathcal{V}, o)$.

The *evaluation* problem asks whether there exists an alternative $o \in \mathcal{X}$ such that $s_w(\mathcal{V}, o) \geq R$.

We apply the 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 LP profiles, the evaluation problem for the Borda rule is in P and it is NP-complete for the three other classes of profiles [Lang *et al.*, 2012]. 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 $k = 2^{p-2}$, the problem is NP-complete, again for all four LP profile types [Lang *et al.*, 2012] (in fact, the result holds for a larger set of values k, we refer for details to Lang et al. [2012]). Clearly, in each case 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 turns out that, as with the k-approval rule, for some values of the parameters, the evaluation problem is NP-complete. We describe one such case here. Namely, we assume two non-negative integers a and b, with $a > b$, as the scoring values and arrange them into 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-2}. We note that if $a = 1$ and $b = 0$ our rule reduces to 2^{p-2}-approval. And we name our rule $2K$-approval if $a = 2$ and $b = 1$.

Theorem 1. *Let w be the scoring vector as described above. The problem to decide for a given UI-UP profile $\mathcal{V}$ and an integer R whether there is an alternative o such that $s_w(\mathcal{V}, o) \geq R$ is NP-complete.*

Proof. We can guess in polynomial time an alternative $o \in \mathcal{X}$ and verify in polynomial time that $S_w(\mathcal{V}, o) \geq R$. So membership in NP follows. Hardness follows from a polynomial reduction from the NP-complete problem *2-MINSAT* [Kohli *et al.*, 1994][1]. Given an instance of 2-MINSAT $I = \langle \Phi, l \rangle$, we construct the set of issues X, the set of alternatives $\mathcal{X}$, the profile $\mathcal{V}$ and the threshold R.

Important observations are that o is among the first quarter of alternatives in an LP tree $\mathcal{L}$ *iff* both the top two most important issues in $\mathcal{L}$ are assigned the preferred values, and that o is among the second quarter *iff* the most important issue is assigned the preferred value and the second most important one is assigned the non-preferred value.

(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 $a + b$ UI-UP LP trees. Assume there is $c_i = \neg X_j \vee X_k \in \Phi$. Then we have $\neg c_i = X_j \wedge \neg X_k \in \Psi$. Firstly, we build $a - b$ duplicate trees such that each one has X_j at the root with preference $1_j > 0_j$, X_k as the second most important issue with $0_k > 1_k$ and other issues in arbitrary order. Secondly, we construct b duplicate trees such that each tree has X_k at the root with preference $1_k > 0_k$, X_j as the second most important issues with $0_j > 1_j$ and other issues in arbitrary order. Thirdly, we build another b duplicate trees such that each vote puts X_k at the root with preference $0_k > 1_k$, X_j as the second most important issues with $0_j > 1_j$ and other issues in arbitrary order. If $b = 0$, we only build $a - b$ duplicate trees of the first type. Denote by $\mathcal{V}_i$ the set of $a + b$ UI-UP LP 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 *iff* there is an assignment satisfying at least $n - l$ formulas in Ψ *iff* there is an alternative scoring at least R with respect to profile $\mathcal{V}$ and $(2^{p-2}, 2^{p-2})$-approval. □

The 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.

4 The Problems in ASP

The winner and the evaluation problems are in general intractable in the setting we consider here. Yet, they arise in practice and computational tools to handle them are needed. Our approach is to develop and evaluate a computational method based on answer-set programming (ASP) [Marek and Truszczynski, 1999]. To this end 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 used in experiments: *clingo* [Gebser *et al.*, 2011], and *clingcon* [Ostrowski and Schaub, 2012]. The encodings 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 the 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 $\mathcal{L}_v$ 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

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

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

[1] The 2-MINSAT problem is defined as follows. Given a set Φ of n 2-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.

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 that alternative takes us to a node in $\mathcal{L}_v$ 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 $\mathcal{L}_1$ in Figure 1 is translated into the logic program in Figure 2 (*voteID(v)* identifies the id of the vote (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 2: Translation of $\mathcal{L}_1$ 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 whether it meets or exceeds the threshold. We first show the encoding of the Borda evaluation problem in *clingo* (cf. Figure 3).

```
1  issue(1..3).
2  numIss(3).
3  val(0..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 3: Borda Evaluation Problem Encoding in *clingo*

Line 5 generates the search space of all alternatives, over three binary issues ("three" and other numeric values are picked to be concrete, the encoding does not change if other arbitrary constants are used).

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 $\mathcal{L}_v$ 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, 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. Formally, the Borda score $s_B(\mathcal{L}_v, o)$ of alternative o in vote $\mathcal{L}_v$ is given by

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

```
1  $domain(1..4).
2  issue(1..3).
3  numIss(3).
4  val(0..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 4: Borda Evaluation Problem Encoding using *clingcon*

where $f(a_i, x_i)$ returns 1 if $a_i = x_i$, 0 otherwise. The Borda score of o with regard to a profile $\mathcal{V}$, is given by

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

In the program in Figure 3, 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*.

The encoding for the winner problem Borda for *clingo* replaces rules 7 and 8 in 3 with the following one rule:

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

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

Encoding the Borda evaluation problem in *clingcon*

The hybrid ASP tool *clingcon* extends ASP with techniques for non-Boolean constraints from Constraint Programming (CP). Its input language allows us to describe logic rules in the same syntax as *clingo*. Moreover, *clingcon* enables us to specify constraints over constraint variables solved by the combined CP solver *Gecode* [Schulte *et al.*, 2010]. In Figure 4 we encode the Borda evaluation problem in *clingcon*.

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 rule 10 in Figure 4 with the following:

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

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

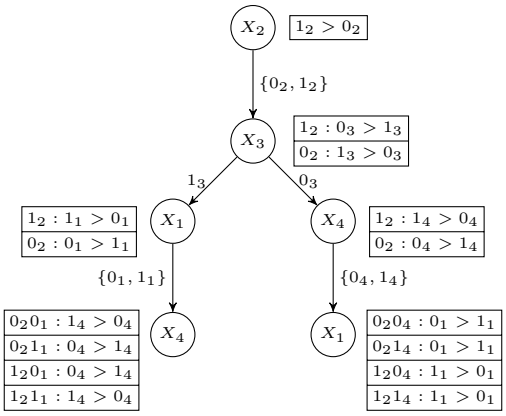


Figure 5: A CI-CP Tree of 4 Issues

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; the splitting node appears 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*. Figure 5 depicts a CI-CP tree in this class.

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.

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 as 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.

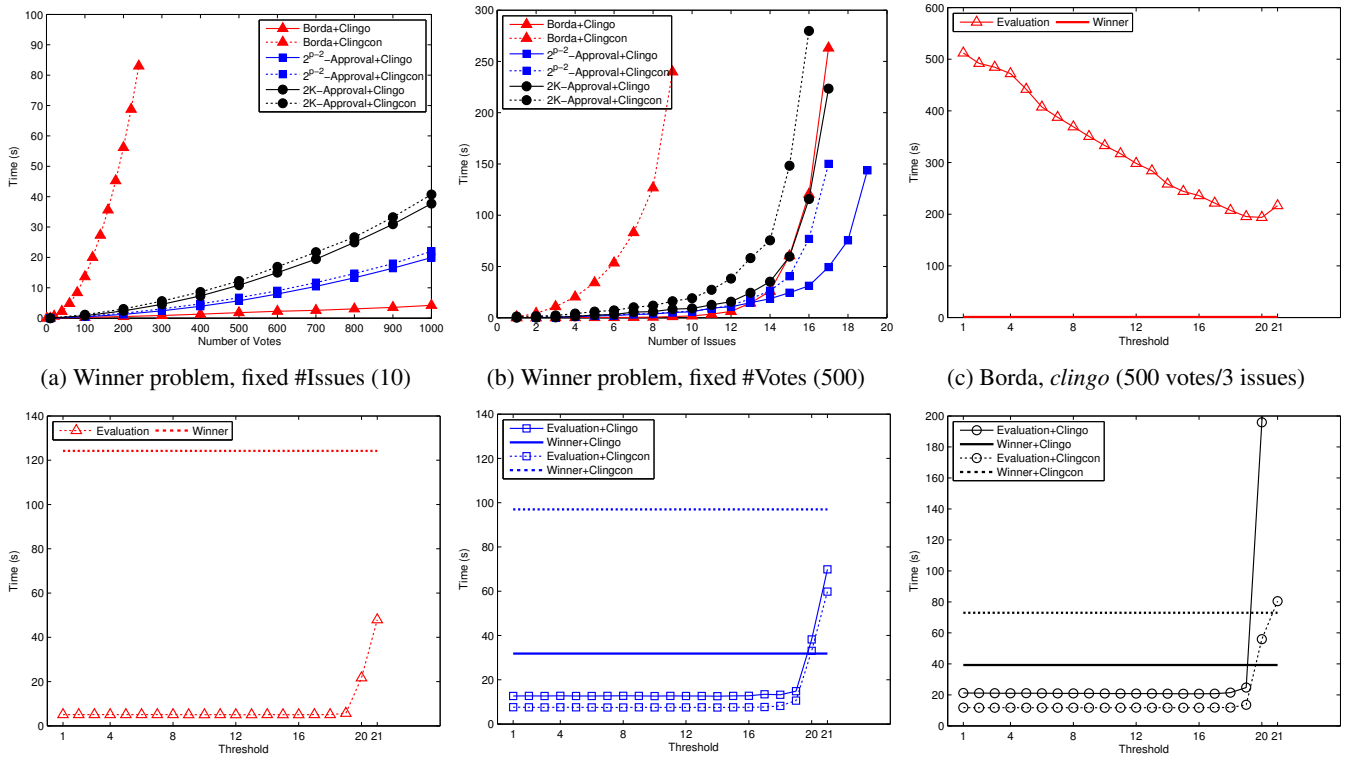
Varying the number of issues and the number of votes
Our experiments on the winner problem for the three voting rules with the fixed number of issues are consistent with the property that the problem is solvable in polynomial time. Both *clingo* and *clingcon* scale up well. Figure 6a depicts the results 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 6b), 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 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 iff* the score of the winner equals or exceeds the threshold. Thus, theoretically, the evaluation problem is at most as complex as the winner problem. Our results confirm that observation except in the case of Borda solved with *clingo*. We compared the two problems by first solving the winner problem, and then by 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 evaluation problem (more precisely, the effectiveness of solvers) when the hardness of the evaluation problem increases with the growing threshold.

First, we note that for *clingo*, the evaluation problem is harder than the winner problem in the entire range for Borda (Figure 6c, experiments with more than 3 issues exceeds the timeout of 10 minutes we used), and for the two other rules, when the threshold is close to the winner's score or exceeds it (Figures 6e and 6f). 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

[2]The restrictions keep the size of generated LP trees linear in the number of issues.



(a) Winner problem, fixed #Issues (10)

(b) Winner problem, fixed #Votes (500)

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

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

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

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

Figure 6: Aggregating *simple* LP trees

an optimization construct, which allows us to keep the size of the ground theory low.

For *clingcon* the situation is different. Figures 6d, 6e and 6f show that the evaluation problem is easier than the winner problem (with just one exception, showing the optimality of the winning score requiring more time than the winner problem). That 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, while *clingcon* is able to avoid it.

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 positional scoring rules is far from being fully understood. First results on the topic were provided by Lang et al. [Lang *et al.*, 2012]; 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 two 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 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 avoids grounding 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 [Boutilier *et al.*, 2004] and answer set optimization (ASO) preferences [Brewka *et al.*, 2003].

Acknowledgments

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

References

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

[Boutilier *et al.*, 2004] 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.

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

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

[Gebser *et al.*, 2011] 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.

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

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

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

[Marek and Truszczynski, 1999] 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.

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

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