Aggregating CP-nets with Unfeasible Outcomes

Umberto Grandi, Hang Luo, Nicolas Maudet, Francesca Rossi

Aggregating CP-nets with unfeasible outcomes

Umberto Grandi[1], Hang Luo[2], Nicolas Maudet[2], and Francesca Rossi[1]

[1] University of Padova
umberto.uni@gmail.com, frossi@math.unipd.it
[2] Université Pierre et Marie Curie
nicolas.maudet@lip6.fr, hang.luo@etu.upmc.fr

Abstract. We consider settings where a collection of agents express preferences over a set of candidates with a combinatorial structure via the use of CP-nets, and we need to exploit the information contained in the CP-nets to choose one of the candidates. Moreover, there is a set of constraints which defines the unfeasible candidates, which cannot be the result of the preference aggregation. We propose a method to achieve this which is based on voting, and considers one variable at a time in a sequence. This method has been studied in the literature to aggregate non-constrained CP-nets. Here we generalise it to work with constrained CP-nets, and we study its properties. The constraints are used to leave in the variable domains only the admissible values. This allows the voting steps to return only feasible values. We find conditions of coherence between the preference expressed in the CP nets and the constraints, in order to guarantee that the classical sequential aggregation method always returns a feasible candidate. Even when such conditions are not met, but the constraints defining the unfeasible candidates have a tree structure (or a structure with bounded tree-width), and the collection of CP-nets is O-legal (that is, the dependency graphs of the CP-nets are compatible), we show that our more general voting procedure can be used, and that it is polynomial in the number of features describing the candidates and in the number of voters.

1 Introduction

Preferences are ubiquitous in real life. Often we need to express our preferences over a large set of objects, which has a combinatorial structure and can be described as the Cartesian product of the domains of a set of decision variables. Consider for example a situation where we give our opinion about cars. A car may be described by several features, like the model, the colour, the engine, the shape, and the maker. Each feature may have several possible choices. We may have various models, colours, engine types, shapes, and makers. If the features are modelled by variables and the choices as variable domains, formally a car can be modelled by an assignment of values to such variables.

However, it may be that not all cars are available in the market. For example, while there are red cars and diesel cars, there could be no red diesel car in the current list of available cars. So we are expressing our preferences over all possible cars, but some of them are actually not available.

Often we take group decisions, together with other individuals. For example, a husband and wife could go and look for a car together, each having his/her own preferences

over all cars. To decide on what car to buy, they need to consider the preferences of both and also the possible feasibility constraints, in order to select a car that is on the market and that satisfies their preferences as well as possible.

In this paper we are interested in modeling and studying these scenarios. To model individual's preferences, we use CP-nets [2], a qualitative formalism to express conditional preferential statements. In the car example, using CP-nets we can say that, if the maker is Citroen, we prefer a gasoline engine to a diesel engine. In particular, we study acyclic CP-nets, which have an acyclic dependency structure, since we believe they are expressive enough to model most rational preference orderings. In acyclic CP-nets, it is computationally easy to determine the most preferred object.

We then use constraints to model the feasibility restrictions. Thus we pair CP-nets with a set of constraints, which define the feasible objects, while the CP-net discriminate among the feasible objects via the preferences. Constrained CP-nets have been studied in the literature, and procedures to obtain a feasible undominated outcome have been proposed [3].

A set of constraints [8] compactly models a set of feasible variable assignments. Each constraint usually involves few variables, and the feasible objects are those that satisfy them all. While the task of finding a feasible variable assignment in a generic constraint set is NP-hard, there are classes of constraint problems where this task is computationally easy. One of these classes consists of those constraint sets whose graph (where nodes model variables and arcs model constraints) is a tree, or a graph with bounded tree-width.

We then consider collections of several acyclic CP-nets, as many as the individuals, plus a set of constraints defining the feasible objects. To aggregate such preferences and make sure we return a feasible object, we use voting rules [1] and we define a sequential voting procedure that considers one variable at a time and uses one of the voting rules to decide the "winning value" for that variable. Sequential voting has been already defined for CP-nets with no feasibility constraints [5]. We show that the classical sequential voting procedure can be used also in this more general setting when CP-nets and constraints are coherent according to three consistency notions that we introduce. In these cases, this procedure returns a feasible outcome. When consistency does not hold, we define a generalization of the classical sequential voting procedure which takes the constraints into account and still returns a feasible outcome. This is computed in time polynomial in the size of the profile, if all the used voting rules are polynomial for winner determination and if the constraints are tractable. This means that the presence of feasibility does not make sequential preference aggregation over CP-nets more difficult, provided that the constraints belong to a tractable class.

2 Background

2.1 CP-nets

CP-nets [2] are a graphical model for compactly representing conditional and qualitative preference relations. CP-nets are sets of *ceteris paribus (cp)* preference statements. For instance, the statement *"I prefer red wine to white wine if meat is served."* asserts that,

given two meals that differ *only* in the kind of wine served *and* both containing meat, the meal with red wine is preferable to the meal with white wine.

A CP-net has a set of features, modelled by variables $F = \{x_1, \ldots, x_m\}$, with finite domains $\mathcal{D}(x_1), \ldots, \mathcal{D}(x_m)$. For each feature x_i, it is given a set of *parent* features $Pa(x_i)$ that can affect the preferences over the values of x_i. This defines a *dependency graph* in which each node x_i has $Pa(x_i)$ as its immediate predecessors. Given this structural information, the agent explicitly specifies her preference over the values of x_i for *each complete assignment* on $Pa(x_i)$. This preference is assumed to take the form of total or partial order over $\mathcal{D}(x_i)$. An *acyclic* CP-net is one in which this dependency graph is acyclic. A *separable* CP-net is one in which there is no preferential dependency (that is, the dependency graph has no edges).

Consider a CP-net whose features are A, B, C, and D, with binary domains containing f and $\overline{f}$ if F is the name of the feature, and with preference statements as follows: $a \succ \overline{a}$, $b \succ \overline{b}$, $(a \wedge b) \vee (\overline{a} \wedge \overline{b}) : c \succ \overline{c}$, $(a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c$, $c : d \succ \overline{d}$, $\overline{c} : \overline{d} \succ d$. Here, statement $a \succ \overline{a}$ represents the unconditional preference for $A = a$ over $A = \overline{a}$, while statement $c : d \succ \overline{d}$ states that $D = d$ is preferred to $D = \overline{d}$, given that $C = c$.

A *worsening flip* is a change in the value of a variable to a value which is less preferred by the CP-statement for that variable. For example, in the CP-net above, passing from $abcd$ to $ab\overline{c}d$ is a worsening flip since c is better than $\overline{c}$ given a and b. One outcome α is *better* than another outcome β (written $\alpha \succ \beta$) iff there is a chain of worsening flips from α to β. This definition induces a preorder over the outcomes.

In general, finding the optimal outcome of a CP-net is NP-hard. However, in acyclic CP-nets, there is only one optimal outcome and this can be found in linear time. We simply sweep through the CP-net, following the arrows in the dependency graph and assigning at each step the most preferred value in the preference table. For instance, in the CP-net above, we would choose $A = a$ and $B = b$, then $C = c$ and then $D = d$. The optimal outcome is therefore $abcd$. Determining if one outcome is better than another (a dominance query) is NP-hard even for acyclic CP-nets. Whilst tractable special cases exist, there are also acyclic CP-nets in which there are exponentially long chains of worsening flips between two outcomes. In the CP-net of the example, $\overline{a}b\overline{c}\overline{d}$ is worse than $abcd$.

2.2 Constraints

In this paper we refer to the usual notation and terminology for constraint satisfaction problems (CSPs), that can be found for example in [8]. In a constraint problem (CSP) (V, D, C), where $V = (v_1, \ldots, v_n)$ is a set of variables, with domains $D = (D_1, \ldots, D_n)$, and C is a set of constraints, a solution is an assignment of values to all its variables that satisfies all constraints. In the following, we say that a partial assignment to a subset of the variables in V is *feasible* if it can be extended to a solution.

We will also use classical results about tractability of constraint problems and the relationship between local and global consistency. In particular, we will exploit the fact that CSPs whose constraint graph has a bounded tree-width, such as trees, are tractable. For trees, a standard polynomial algorithm to solve them involves deciding on an ordering over which to instantiate the variables to construct a solution, achieving

directional arc-consistency on such an order, and then instantiating the variables with no backtracking.

Another tractability result that we will use in this paper is the fact that, in CSPs with Boolean variables and binary constraints, strong 3-consistency (that is, arc- and path-consistency) is sufficient to assure global consistency. This means that it is polynomial to solve such CSPs. In fact, even if we do not have strong 3- consistency, we can achieve it in polynomial time while remaining in the class of binary constraints with Boolean variables, and then we can apply this result.

We will also use a graph which is obtained by the constraint graph of a CSP by first achieving path-consistency and then taking only those edges that correspond to non-trivial constraints (that is, constraints that are a proper subset of the Cartesian domain of their variables). We will call this the *path-closure* graph of the CSP.

2.3 Constrained CP-nets

A constrained CP-net is just a CP-net N with the addition of a set of constraints C over the same variables. An outcome is feasible if it satisfies all constraints in C. An optimal outcome for a constrained CP-net (N, C) is a feasible outcome which is not dominated by any other feasible outcome in the CP-net preference ordering.

While for acyclic CP-nets finding an optimal outcome is computationally easy, for acyclic constrained CP-nets it is as difficult as solving (possibly several times) the constraint set C. In [3] an algorithm (Search-CP) is defined to find an optimal outcome in a constrained CP-net.

Therefore, when the constraint set is tractable, for example it has a tree structure, then this problem is computationally easy. In this paper we consider constrained CP-nets where the CP-net is acyclic, and for some results we will also exploit the tractability of the constraint set. However, we will not compute the optimal outcome of any single constrained CP-net, but rather use the constraints to make sure we obtain a feasible outcome as the result of the aggregation of several CP-nets.

Even if the literature has focussed on finding optimal outcomes of constrained CP-nets, it is also an interesting question to know whether preferences expressed by the CP-net comply (and to what extent) with the constraints. In Section 3 we discuss several possible ways to define such a compliance.

2.4 Voting theory

A voting rule allows a set of voters to choose one among a set of candidates. Voters need to submit their vote, that is, their preference ordering over the set of candidates (or part of it), and the voting rule aggregates such votes to yield a final result, usually called the winner. In the classical setting [1], given a set of candidates, a *profile* is a collection of total orderings over the set of candidates, one for each voter. Given a profile, a *voting rule* maps it into a single winning candidate.

Some examples of widely used voting rules are:

- *Plurality*, where each voter states who the preferred candidate is, and the candidate who is preferred by the largest number of voters wins;

– *Borda*, where, given m candidates, each voter gives a ranking of all candidates and the i^{th} ranked candidate scores $m - i$, and the candidate with the greatest sum of scores wins;
– *Approval*, where each voter approves between 1 and $m - 1$ candidates on m total candidates, and the candidate with most approvals wins;
– *Copeland*, where the winner is the candidate that wins the most pairwise competitions against all the other candidates.

When there are ties, a unique winner is chosen according to some tie-breaking rule.

Voting theory has considered many desirable properties of voting rules. Some examples are anonymity, neutrality, consistency, monotonicity, Pareto efficiency, independence of irrelevant alternatives (IIA), non-dictatoriality, strategy proofness, and participation [1]. All the above rules are anonymous, neutral, non-dictatorial, monotone, and Pareto efficient, while only Approval is IIA. All but Copeland are consistent and participative.

2.5 Aggregating CP-nets

There is extensive literature that has considered the task of aggregating preferences modelled via CP-nets [5]. In this setting, n agents express their preferences over a set of candidates with a combinatorial structure: there are m features, and each candidate is an assignment of values to all features. Agents' preferences over the candidates are usually modeled via acyclic CP-nets. Moreover, the dependency graphs of such CP-nets are usually assumed to be compatible with a linear order O over the features: for each voter, the preference over a feature is independent of features following it in O.[3] This implies that the n CP-nets $N_1, \ldots, N_n$ are such that the union of their dependency graphs, that we call $Dep(N_1, \ldots, N_n)$, does not contain cycles. Notice that CP-nets with this property may have different dependency graphs.

Given n agents, m binary features, and a linear ordering O over the features, a profile is thus a collection of n acyclic CP-nets over the m features which are compatible with O.

To aggregate the preferences expressed via CP-nets, what is often done is to employ a sequential approach, with as many steps as the number of features. The algorithm uses as many voting rules as the number of features, say $\langle r_1, \ldots, r_m \rangle$.

Taken the features in the order O, say $\langle x_1, \ldots, x_m \rangle$, the aggregation considers one feature at a time in this order and returns a "winner", that is, a variable assignment for such variables, say $\langle d_1, \ldots, d_n \rangle$. It starts from variable x_1, which is for sure an independent variable, and applies voting rule r_1 to the profile obtained by the preference orderings given by all CP-nets over the domain of x_1, returning a winner value d_1. If we have already considered variables $x_1, \ldots, x_k$, obtaining values $d_1, \ldots, d_k$, we then consider variable x_{k+1} and apply voting rule r_{k+1} to the profile obtained by considering the preference ordering over the domain of x_{k+1} in all CP-nets. If x_{k+1} is a dependent variable, we choose the preference ordering which corresponds to the assignment of the previous variables. Notice that all parents of x_{k+1} must have been assigned already at

[3] This coincides with the notion of O-legality in [5].

this point, because of the way the ordering O is defined. This procedure is sometimes called sequential voting [5], or also Level Aggregation (LA) [7].

Example. Consider three agents, each expressing their preferences over candidates defined by 3 binary features. So we have 3 CP-nets N_1, N_2, and N_3, with features A, B, and C, where each feature X has values x and $\bar{x}$. N_1 contains the preferential statements $a \succ \overline{a}$, $b \succ \overline{b}$, $(a \wedge b) \vee (\overline{a} \wedge \overline{b}) : c \succ \overline{c}$, $(a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c$. We recall that $a \succ \overline{a}$ represents the unconditional preference for $A = a$ over $A = \overline{a}$, while $(a \wedge \overline{b}) \vee (\overline{a} \wedge b) : \overline{c} \succ c$ states that $C = c$ is preferred to $C = \overline{c}$, when $A = a$ and $B = \overline{b}$ and also when $A = \overline{a}$ and $B = b$. Thus, in N_1, A and B are independent variables, while C depends on both A and B. N_2 contains instead the following preferential statements: $a \succ \overline{a}$, $a : b \succ \overline{b}$, $\overline{a} : \overline{b} \succ b$, $b : c \succ \overline{c}$, $\overline{b} : \overline{c} \succ c$. Thus, in N_2, A is an independent variable, while B depends on A and C depends on B. N_3 is defined by: $a \succ \overline{a}$, $b \succ \overline{b}$, $c \succ \overline{c}$. Thus, in N_3, all variables are independent. Figure 1 shows this profile.

Consider now ordering $O = \langle A, B, C \rangle$, and let us apply the sequential voting procedure with voting rule Majority for all three variables. For variable A, we have the preferences $a \succ \overline{a}$ from all agents, thus we select $A = a$. Then, given this choice, we pass on to variable B, getting preferences $b \succ \overline{b}$ from all agents. Thus we choose $B = b$. Notice that, while feature B is independent from A in agent 1 and 3, in agent 2 it depends on A. Thus the preferences on the values of B in such an agent are those corresponding to the value of A chosen in the previous step. Passing on to C, we get preferences $c \succ \overline{c}$ from all agents, thus we choose $C = c$. Thus the sequential procedure chooses the variable assignment $\langle A = a, B = b, C = c \rangle$.

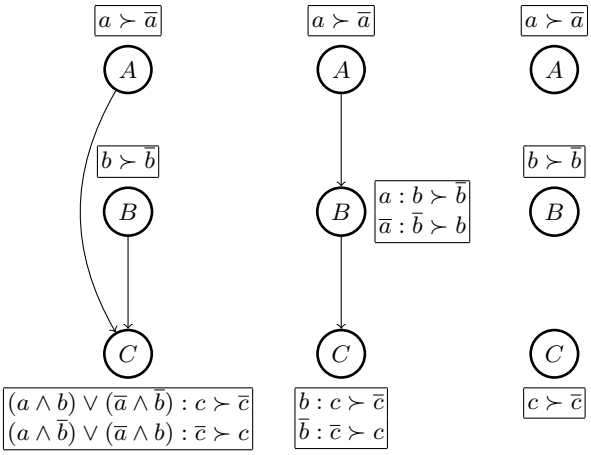

Fig. 1. A profile of CP-nets.

Sequential voting provides an efficient way to determine the winning candidate when preferences are expressed compactly with CP-nets or other preference formalisms. Moreover, it maintains many of the desirable properties of the local voting rules used at every step [5].

3 Consistency in constrained CP-nets

Given a constrained CP-net (N, C), we now study some notions of consistency between the preference structure expressed by N and the set of constraints C. The reason we are interested in these consistency notions is that in some cases they make the aggregation simpler, when preferences are expressed by a collection of constrained CP-nets, as we will see in Section 5.

3.1 Consistency notions

The first notion of consistency relates the optimal outcome of the CP-net to the constraints.

Definition 1. *A constrained CP-net* (N, C) *is* top-consistent *if the optimal outcome of* N *satisfies the constraints in* C*.*

For example, the CP-net of agent 1 in Figure 1 is top-consistent with the set of constraints $\{A = B\}$.

The next notion of consistency acts at the variable level and makes sure that feasibility is maintained when passing from the parents of the variable to its most preferred value.

Definition 2. *A constrained CP-net* (N, C) *is* locally-consistent *if there is no line in the CP-tables of* N *of the form* $o : b > \bar{b}$ *such that* o *is feasible but* ob *is not.*

Since o and ob can be partial outcomes, that is, assigning values to only some of the variables, we recall that a partial outcome is feasible if it can be extended to a solution.

The third notion of consistency we define is a structural property, that related the dependency graph of the CP-net to the path-closure graph of the constraints.

Definition 3. *A constrained CP-net* (N, C) *is* dependency-consistent *if the path-closure graph of* C *is a subset of the undirected version of the dependency graph of* N*.*

Dependency consistency can be natural in several settings. For example, if constraints are known in advance, the process of specifying a CP-net will exploit preferential dependencies among variables connected by a constraint to express qualitative preferences over the partial outcomes over such variables.

Theorem 1. *If a constrained CP-net is both locally and dependency-consistent, then it is also top-consistent.*

Proof. If we have both local and dependency consistency, the optimal outcome is feasible (that is, we have top consistency). In fact, let us compute the optimal outcome by instantiating one variable at a time, in an order which is compatible with the dependency graph of the CP-net (that is, parents come before their children). We start from the independent variables and we give them their most preferred value. This is a feasible partial assignment since, by dependency consistency, there are no constraints among independent variables. At any step, we instantiate a new variable to its most preferred value given the chosen instantiation of its parents. If the partial assignment before this step was feasible, also the new partial assignment is feasible because of local and dependency consistency. Thus all partial assignments built during the procedure, included the last one which is the optimal outcome, are all feasible. A feasible complete assignment is, by definition, a solution. □

It is easy to see that neither local nor dependency consistency alone imply top consistency, and viceversa.

Example. Consider the CP-nets in Figure 1 and the constraints $c_{AB} = \{(A = a, B = b), (A = \overline{a}, B = \overline{b})\}$ *and* $c_{BC} = \{B = b, C = \overline{c}), (B = \overline{b}, C = c)\}$. *None of the CP-nets are top consistent. Moreover,* N_1 *is not locally consistent because of the CP-table for feature* C*:* ab *is a partially feasible assignment but* abc *is not.* N_2 *is not locally-consistent either, again because of the CP-table for* C*:* b *is feasible but* bc *is not. On the other hand,* n_3 *is locally consistent. Only* N_2 *is dependency consistent.*

3.2 Checking the consistency notions

We now study the computational complexity of checking the above three notions of consistency in a constrained CP-net. In what follows we assume CP-nets to be acyclic.

Theorem 2. *Given a constrained CP-net* (N, C)*, it is polynomial to check whether it is top-consistent or dependency consistent.*

Proof. For top consistency, it is sufficient to find the optimal outcome of N and check whether it satisfies the constraints in C. Since N is acyclic, this is computationally easy.

For dependency consistency, we just need to compare the dependency graph and the path-closure graph of the constraints. Once we have the two graphs, this is linear in their size. The path-closure graph can be obtained by achieving 3-consistency on the constraints, which is polynomial. □

Theorem 3. *Given a constrained CP-net* (N, C) *with Boolean variables, with* C *a set of binary constraints, it is polynomial to check whether it is locally consistent.*

Proof. We need to check that ob is feasible, for each row in the CP-tables of the form $o : b > \overline{b}$ such that o is feasible. Since constraints are binary, for each row in a CP-table, this can be done in polynomial time. In fact, checking that a partial outcome is feasible with a set of binary constraints is computationally easy if variables are Boolean (it amounts to solving a 2SAT problem). The number of rows in a CP-net may be exponential in the number of issues, but not in the size of the CP-net which is given in the input. Thus the overall complexity is polynomial. □

Observe that local consistency in general cannot be checked in polynomial time if constraints are not binary, even if variables are Boolean, since it would require solving a SAT problem, while in the binary case it is 2-SAT.

3.3 Achieving top and local consistency in constrained CP-nets

Assume that (N, C) is a constrained CP-net which is not top-consistent or not locally-consistent. This can happen in scenarios in which we have our own preferences over the outcomes expressed via a CP-net, and somebody gives us the constraints describing the feasible outcomes, and the two things together do not have the desired notion of consistency. We would like to modify our CP-net as little as possible in order to obtain either top consistency or local consistency.

Top consistency. To achieve top consistency, we may adopt the following procedure. Let us start from any independent variable (there must be one since N is acyclic) and have one step for each variable, in an order which is compatible with the dependency graph of the CP-net (parents come before their children), computing the optimal outcome. If at any step j, the partial outcome o obtained so far is not feasible, then we modify the row of the CP-table of variable x_j corresponding to the parents' assignment in o doing a switching of the ordering. This assures that the new partial outcome is feasible. This algorithm will produce in polynomial time a CP-net which is top-consistent if the constraints are binary. However, it does not assure that the resulting CP-net is minimally distant from the given one, if the distance is the number of different orderings in the CP-tables. However we conjecture it would be computationally difficult to find the one which is minimally distant.

Local consistency. Instead, to obtain a CP-net which is locally-consistent, it is sufficient to check each row in the CP-tables for the condition of local consistency, again following an order of the variables which is compatible with the dependency graph. If one of the rows fails the consistency check, then the preference expressed in this row needs to be inverted. Since we are moving forward following the dependency structure of N, we are guaranteed that one of the two possible orders in a row of the CP-table must be consistent. Notice that this algorithm is different from the previous one since we need to check all rows of the CP-tables and not just those involved in the computation of the optimal outcome. Again, the assumption of binary constraints is crucial for this algorithm to be polynomial. Unlike the previous one, this algorithm guarantees that the resulting CP-net is minimally distant from the given one, if the distance is the number of different orderings in the CP-tables.

4 Constrained Profiles

A constrained profile models the scenario in which we have several individuals who express their preferences over a common set of outcomes by using CP-nets, and the constraints model the set of feasible outcomes. Only those outcomes that satisfy all

constraints can be returned as the result of the aggregation of the preferences of the individuals.

Formally, a *constrained profile* is a collection of CP-nets $\{N_1, \ldots, N_n\}$ plus a set of constraints C. This can also be seen as a collection of constrained CP-nets $\{(N_1, C), \ldots, (N_n, C)\}$, all having the same constraints.

Notice that all CP-nets share the same set of feasible (and thus unfeasible) candidates, which are those defined by C. Moreover, the CP-nets of all agents share also the variables and the variable domains. So, what can be different in two agents is the dependency graph of their CP-nets, as well as CP-tables of the CP-nets.

In this paper, we restrict our attention to constrained profiles which are O-legal, that is, there is an ordering O of the variables such that, for each CP-net, the preference over a feature is independent of features following it in O.

Notice that O-legality implies that all CP-nets in the constrained profile are acyclic.

Example. As an example of a constrained profile, let us consider the CP-nets in Figure 1, with the addition of the set of constraints $c_{AB} = \{(A = a, B = b), (A = \overline{a}, B = \overline{b})\}$, $C_{BC} = \{B = b, C = \overline{c}), (B = \overline{b}, C = c)\}$. *It is easy to see that this profile is* O*-legal: there is an ordering* O *of the variables which is compatible with all dependency links, namely* $O = (A, B, C)$. *Observe that the top outcome is* abc *for all three agents, and this would be the result of sequential majority over the three CP-nets. However, this outcome is not feasible (only* $ab\overline{c}$ *and* $\overline{a}\overline{b}c$ *are).*

5 Aggregating preferences in constrained profiles

The goal is to take a constrained profile and return a feasible outcome, which should satisfy the preferences of the individual CP-nets as much as possible. As we know, when we have no constraints on the feasible outcomes, sequential voting is used to perform such an aggregation. We will now see that sometimes sequential voting is all we need also in presence of constraints. In general, however, we need to take constraints into account. This can be done by adapting the sequential voting procedure, while maintaining a polynomial time complexity if constraints are tractable.

5.1 Top, local, and dependency consistency

Under assumptions the consistency notions introduced in Section 3, sequential aggregation using the majority rule outputs a feasible outcome. The first result applies when we have top consistency, but requires CP-nets to be separable, that is, to have no dependency structure.

Theorem 4. *If* $\langle (N_1, \ldots, N_n), C \rangle$ *is a constrained profile such that all* N_i *are top-consistent and separable, and* C *is a set of binary constraints, then the winner determined by sequential voting with the majority rule is feasible.*

Proof. Since all N_i are separable (and variables are binary), then the order followed by sequential majority is irrelevant, and the problem is equivalent to binary aggregation in which all individuals submit their top outcome and issue-by-issue majority voting

is used. We can therefore use the following result from the binary aggregation literature: issue-by-issue majority outputs a feasible outcome given feasible input (that is, it is collectively rational) if and only if the constraints are equivalent to a conjunction of disjunctions of size 2 [4, 6]. First we observe that, by top consistency, each individual top outcome satisfies the constraints. Second, since we assume constraints in C to be binary, each constraint can be written as a conjunction of disjunctions of size 2, thus the whole set of constraints can also be written in this way. Therefore this result applies here. If the constraints are not binary, it is possible to find examples in which the outcome of sequential majority is not feasible. □

When the CP-nets have a non-empty dependency structure, we can still apply standard sequential voting to get a feasible outcome if they are both locally and dependency consistent (and thus also top consistent). So we need a stronger property on the CP-nets when we have preferential dependencies.

Theorem 5. *If $\langle (N_1, \ldots, N_n), C \rangle$ is a constrained profile such that all N_i are locally-consistent and dependency-consistent, and C is a set of binary constraints, then the winner determined by sequential voting with the majority rule is feasible.*

Proof. We will prove by induction on the number of variables that, at each step i between 1 and m, the partial assignment generated until step i is feasible. For step 1, it is trivially true since the CP-nets are locally consistent, so the most preferred value in an independent variable must be feasible. This means that all CP-nets vote for a feasible value for the first variable, and thus majority chooses a feasible value.

Let us assume that the statement is true until step i, and let us consider step $i+1$. We have a feasible partial assignment $\langle v_1, \ldots, v_i \rangle$ obtained so far. For variable $i+1$, assume that there is a majority in favor of b, i.e., at least a majority of the individual CP-nets prefer b to $\bar{b}$ given the partial assignment obtained so far. This means that if individual j is part of this majority, then N_j contains the row $o_j : b > \bar{b}$, where o_j is the assignment of the parent variables of variable i in CP-net N_j which occurs in the current feasible assignment. By dependency consistency we know that the parent variables of variable i in each individual CP-net include all variables k that are related with i by a constraint. By local consistency, we also know that $o_j b$ is feasible for each j between 1 and n. Thus also $\langle v_1, \ldots, v_i, b \rangle$ is feasible.

Therefore all partial assignments generated during the sequential voting procedure are feasible, including the last one, which is a complete assignment and thus a solution of all the constraints. □

5.2 Aggregation in non-consistent profiles

When none of the sufficient conditions mentioned above hold, we can obtain a feasible outcome by modifying the sequential voting procedure to take the constraints into account. Starting from the LA procedure already defined in the literature to aggregate CP-nets, we define the procedure CLA, for Constrained LA procedure. CLA is very similar to LA, except that it will work on possibly reduced variable domains, because of the constraints. As each step, the constraints will tell us what domain values to consider, in order to get a feasible outcome.

Algorithm 1: CLA

Input: A constrained profile $\langle (N_1, \ldots, N_m), C \rangle$, n voting rules $r_1, \ldots, r_n$, an ordering $O = \langle x_1, \ldots, x_n \rangle$
Output: a variable assignment $\langle x_1 = v_1, \ldots, x_n = v_n \rangle$
for $i = 1$ *to* n **do**
  T_i = the constraint graph of C (a tree), rooted at x_i;
  C' = DAC(T_i);
  D_i = the domain of x_i in C';
  if $D_i = \emptyset$ **then**
    return *No feasible candidate*
  for $j = 1$ *to* m **do**
    o_j = the ordering over D_i given by the CP-table in N_j for $x_1 = v_1, \ldots, x_{i-1} = v_{i-1}$;
    o'_j = o_j restricted to D'_i;
  $v_i = r_i(o'_1, \ldots, o'_n)$;
  Add the constraint $x_i = v_i$ to C;
return $\langle x_1 = v_1, \ldots, x_n = v_n \rangle$

The first thing we need to do is to preprocess the constraints in C so to bring to the variable domains the information about the feasible candidates. In fact, since LA is a sequential voting procedure which considers one variable at a time, it is important to leave in the domain of each variable only the values that belong to feasible candidates.

As in the classical sequential voting procedure, we have a collection of m voting rules $\langle r_1, \ldots, r_m \rangle$ that will be used in the m steps of the procedure, one step for each variable. If variables are Boolean, of course all r_i will be the majority voting rule. Assume for now that the constraint set has a bounded tree-width, so it belongs to a tractable class. For sake of easiness of presentation, let us consider a tree- like shape. However, the CLA procedure works also for bounded tree-width constraint sets.

Since the constraints have a tree shape, it is indeed possible to leave in the domain of each variable only those values that appear in some feasible candidate. We just need to consider the variable ordering O, take the first variable x_1, use it as the root of the tree, and achieve directional arc-consistency to this tree. At the end, the new domain of x_1, say D'_1, will contain only those values that appear in some feasible candidate. We can now apply the voting rule r_1 to the profile over variable x_1, where however the domain of x_1 has been reduced to D'_1. This will choose a value for x_1, say v_1, which is feasible (that is, it can be extended to a solution).

Let us now pass to the second variable x_2. Given the value v_1 chosen for x_1, we set $x_1 = v_1$ in C and in all the CP-nets and we apply again DAC bottom-up, now by using x_2 as the root of the tree. This will generate a new domain for x_2, say D'_2, which will contain only those values that appear in some feasible candidate. We can now apply the voting rule r_2 to the profile over variable x_2 (given $x_1 = v_1$), where however the domain of x_2 has been reduced to D'_2. This will choose a value for x_2, say v_2. Now we have the partial assignment $\langle x_1 = v_1, x_2 = v_2 \rangle$, which is feasible.

We then continue like this until all variables have been assigned. The winning candidate is then $\{x_1 = v_1, \ldots, x_n = v_n\}$.

Since C is a tree, the first application of DAC will tell us if there are feasible candidates. If no variable domain is empty after the first DAC, then we know there is at least one feasible candidate, and the later applications of the DAC procedure will never generate any empty variable domain.

Example. As an example of the application of the CLA algorithm to a constrained profile, consider again the constrained profile in Figure 1, with ordering $O = \{A, B, C\}$ *and constraints* $c_{AB} = \{(A = a, B = b), (A = \overline{a}, B = \overline{b})\}$, $c_{BC} = \{B = b, C = \overline{c}), (B = \overline{b}, C = c)\}$. *Since we do not have top consistency, nor local and dependency consistency, for all CP-nets, we cannot use classical sequential voting to select a feasible outcome. We will thus use the CLA procedure. Since the variables are binary, we use majority voting at each step. CLA first achieves DAC to the constraint set, which is a tree, rooted at* A. *This removes the value* $\overline{b}$ *from the domain of* B, *because of the constraint* c_{BC}, *and it also removes the value* $\overline{a}$ *from the domain of* A, *because of constraint* c_{AB}. *We now apply majority voting to the profile related to variable* A, *getting* $A = a$. *We then add the constraint* $A = a$ *to the initial set of constraints and we pass on to the second variable,* B. *We achieve DAC to the tree rooted at* B, *which does not remove anything from any domain. We apply majority to the profile for* B, *getting* $B = b$, *and we add this as a new constraint. Finally, we achieve DAC on the tree rooted at* C, *leaving only* $\overline{c}$ *in the domain of* C, *and, by majority voting, we get* $C = \overline{c}$. *Thus the result of CLA is* $(A = a, B = b, C = \overline{c})$. *This variable assignment satisfies all constraints. Notice that the outcome of a sequential voting procedure over the same profiles, without the constraints, would be* $(A = a, B = b, C = c)$, *which does not satisfy the constraints.*

On the other hand, if C is not tree-shaped, achieving DAC could leave in the variable domains also values that do not appear in any feasible candidate. Thus, once a value for a variable is chosen, it could be that there is no value for the next variable which is compatible with it. This means that the CLA procedure should backtrack its previous choices (for example the last one made) and replace it with another value. It could also be the case that no feasible candidate exists, and this will result in backtracking over the choices until no more alternative choice is available. Thus the CLA procedure needs to perform search if achieving DAC (or adaptive consistency) does not leave the domains *minimal*, that is, containing only the values that participate in at least a solution.

5.3 Properties of CLA

The most important property to prove is that, in the setting we are considering, CLA always returns feasible outcomes, in time polynomial in the size of the input. We recall that our setting assumes that we have a constrained profile $\langle (N_1, \ldots, N_n), C \rangle$, where C has a tree-like constraint graph, m voting rules $r_1, \ldots, r_m$, and an ordering $O = \langle x_1, \ldots, x_m \rangle$ which makes the profile O-legal.

Theorem 6. *The variable assignment* $\langle x_1 = v_1, \ldots, x_m = v_m \rangle$ *returned by CLA satisfies all constraints in* C.

Proof: Consider the output of CLA, say $\langle v_1, \ldots, v_m \rangle$. Take any constraint in C, say c, between variables x_i and x_j. We need to prove that $\langle x_i = v_i, x_j = v_j \rangle$ satisfies c. At step i, CLA applied DAC to the tree rooted at x_i, restricting the domain of x_i. So, by definition of DAC, v_i is a value for x_i such that there is a value in x_j (and in any other variable) which satisfies c. After doing that, CLA has added the constraint $x_i = v_i$ to C. Then, at step j, CLA applied DAC again, to the tree rooted at x_j, thereby reducing the domain of x_j to only those values that have support in the domains of all variables, thus also in the domain of x_i, which is now containing just the value x_i. Since v_j is in the domain of x_j, this means that $\langle x_i = v_i, x_j = v_j \rangle$ satisfies c. □

Theorem 7. *CLA works in time $O(n \times (md^2 + n + t))$, where m is the number of variables, d is the size of the largest domain among $D_i, \ldots, D_m$, n is the number of agents, and $t = f(n, d)$ is the time complexity for winner determination in the most computational expensive of the voting rules $r_1, \ldots, r_m$.*

Proof: CLA performs at most m steps. At each step, it achieves DAC on a tree with m variables with domain size at most d. This takes $O(md^2)$ time. It then reduces the n orderings over the current variable domain to the new domains computed by DAC. Finally, it applies the voting rule for that step to such orderings. □

It is worth noting that, in the case of just one voter, we have a single constrained CP-net, and CLA returns a feasible outcome which is undominated in the CP-net preference ordering. This is equivalent to what is done in [3]. However, since we consider tree-shaped constraint sets, we can get this outcome in polynomial time. We therefore get this useful result out of our aggregation procedure. Observe moreover that if we start from acyclic CP-nets, an order O that makes the profile O-legal (or the conclusion that there is no such order) can be found in time polynomial in the number of variables. It is indeed sufficient to take the union of all dependency graphs and take any linearisation of it, if there is one. Any such ordering would give the same result of sequential majority, since preferential dependencies are all taken care of in the union graph.

Theorem 8. *Given a constrained CP-net $\langle N, C \rangle$, where N is acyclic and the constraint graph of C is a tree, finding an undominated feasible outcome is in $\mathcal{P}$.*

Proof: By Theorem 6, the variable assignment returned by CLA is feasible, that is, it satisfies all constraints in C.

We have just one CP-net N and a set of constraints C. Thus, every time we use a voting rule r_i, this voting rule acts as the identity, thus returning the top choice in the preference ordering it gets in input. We will prove by induction on i that, after step i, $\langle x_1 = v_1, \ldots, x_i = v_i \rangle$ is undominated by feasible outcome (in the outcome ordering of N restricted to $x_1, \ldots, x_i$) and it satisfies all constraints in C. It is trivial for step 1, since v_1 is the top choice in the restricted domain of x_1, obtained after applying DAC to the tree C rooted at x_1. Assume the statement is true at step i and let us now consider step $i + 1$. CLA returns v_{i+1}, which is the top element of the preference ordering in the restricted domain of x_{i+1}. Since $\langle x_1 = v_1, \ldots, x_i = v_i \rangle$ is an undominated outcome over variables $x_1, \ldots, x_i$, and since x_{i+1} is the top element in the feasible domain of

x_{i+1}, there is no other extension of $\langle x_1 = v_1, \ldots, x_i = v_i \rangle$ to variable x_{i+1} which can be more preferred to $\langle x_1 = v_1, \ldots, x_i = v_i, x_{i+1} = v_{i+1} \rangle$. □

Sequential voting with CP-nets has been studied also from the point of view of the properties which can, or cannot, be transferred from the "local" voting rules $r_1, \ldots, r_n$ to the "global" sequential voting rule LA [5]. It is easy to see that, if a property transfers from local to global in the LA procedure, then it also transfers for CLA. In fact, CLA is just LA but on possibly smaller variable domains. So we are performing a domain restriction on the set of possible profiles. If a property is true of a sequential voting procedure when considering a larger set of profiles, it will remain true when we consider a smaller set. This is true for properties like anonymity, consistency, strong monotonicity, and monotonicity (of r_m), as shown in [5].

If instead a property does not transfer from local to global, then by passing from LA to CLA the same examples showing this still hold. Examples of properties that do not transfer from local to global LA are neutrality, efficiency, and participation (see again [5]). However, it could be that the domain restriction imposed by the constraints removes those profiles which are problematic for that property. We plan to study specific constraint classes that could allow some properties to transfer from local to global for CLA, even though they do not do so for LA.

Theorem 9. *If all voting rules $r_1, \ldots, r_m$ are anonymous (resp., consistent, strong monotone), then so is CLA. If r_m is monotone, then so is CLA. If $r_1, \ldots, r_m$ satisfy neutrality (resp., efficiency, participation), it could be that CLA does not satisfy it.*

Proof: It follows directly from the analogous results in [5] for LA.□

6 Conclusions

Often we need to aggregate preferences of several agents over a set of candidates which satisfy certain requirements. Agents may express preferences over a superset of the feasible candidates, with a combinatorial structure, thus being able to use compact preference modeling frameworks such as CP-nets. But the result of preference aggregation can only be a feasible candidate.

We modeled these settings by defining constrained profiles consisting of a collection of CP-nets and a set of constraints, and by proposing a procedure to return a feasible candidate by interleaving constraint solving and voting. If the CP-nets are acyclic and the constraint set has a tree shape, the procedure, called CLA, requires polynomial time if all the voting rules work in polynomial time. Other properties, but not all, transfer from the voting rules to the CLA procedure.

We also identified conditions on the compliance of the CP-nets to the constraints that allow us to forget about the constraints when aggregating the preferences, and thus using standard sequential voting to get a feasible outcome.

We plan to study several other properties of the CLA procedure, among which the complexity of manipulation, bribery, and control. We also plan to test it experimentally on real life profiles from the preflib.org website, where we will add some constraints.

Acknowledgements

The work of Umberto Grandi and Francesca Rossi was partially supported by the strategic project "KIDNEY - Incorporating patients' preferences in kidney transplant decision protocols" funded by the University of Padova.

References

1. K. J. Arrow and A. K. S. amd K. Suzumura. *Handbook of Social Choice and Welfare.* North-Holland, Elsevier, 2002.
2. C. Boutilier, R. I. Brafman, C. Domshlak, H. 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.
3. C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. Preference-based constrained optimization with cp-nets. *Computational Intelligence*, 20:137–157, 2004.
4. U. Grandi and U. Endriss. Lifting integrity constraints in binary aggregation. *Artificial Intelligence*, 199-200:45–66, 2013.
5. J. Lang and L. Xia. Sequential composition of voting rules in multi-issue domains. *Mathematical social sciences*, 57:304–324, 2009.
6. C. List and C. Puppe. Judgment aggregation: A survey. In *Handbook of Rational and Social Choice*. Oxford University Press, 2009.
7. N. Maudet, M. S. Pini, F. Rossi, and K. B. Venable. Influence and aggregation of preferences over combinatorial domains. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2012)*, pages 1313–1314, 2012.
8. F. Rossi, P. V. Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.