HAL
open science

Dealing with numerical imprecision in mathematical programs for Electre Tri models disaggregation

Olivier Cailloux

To cite this version:

Olivier Cailloux. Dealing with numerical imprecision in mathematical programs for Electre Tri models disaggregation. 2014. hal-00985881

HAL Id: hal-00985881

https://hal.science/hal-00985881

Submitted on 30 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Dealing with numerical imprecision in mathematical programs for Electre Tri models disaggregation

Olivier Cailloux

June 26, 2012

1 Introduction

In this paper, we are interested in decision problems formulated as multicriteria sorting problems, i.e., when a finite number of alternatives from a set A evaluated on a set of criteria $\{g_j, j \in J\}$ are to be assigned to one of k predefined ordered categories $c_1 \ll c_2 \ll ... \ll c_h \ll ... \ll c_k$ (c_1 being the worst category, and c_k the best one). The assignment is done based on the comparison of the alternatives to external norms, rather than by comparison of the alternatives to each other.

Several approaches have been proposed to address such multicriteria sorting problem [8, 5]. We consider a well know multiple criteria sorting method, Electre Tri [4, 7, 9]. More precisely, we consider a variant of the Electre Tri method in line with the axiomatic work of Bouyssou and Marchant [1, 2]. This variant assigns alternatives using the alternatives performances and preferential parameters of three types: profiles defining the category limits, weights specifying the importance of each criterion, and veto thresholds. To support specifying their preferences, we suppose that the DMs are able to provide assignment examples, i.e. alternatives (fictitious or real) associated to the categories the DMs think they belong to. Such assignment examples can correspond to past decision records or be expressed directly by DMs.

Most of the work on preference elicitation in Multicriteria Decision Aid focuses on representing the preferences of a single decision maker (DM). We are interested in elicitation procedures for multiple DMs that make it possible for each DM to provide individual preference information in order to build a multiple criteria sorting model accepted by each DM as representing the group preferences. We present linear programs able to find, on the basis of assignment examples provided by DMs, common profiles, shared among all the DMs, but allowing their weights (criteria importance factors) to vary individually. This can be used as a first step towards reaching an agreement on a preference model.

In a recent paper [3], three linear and mixed integer programs solving the following problems are described.

ICL, or Infer Category Limits, finds, if possible, a set of profiles such that it is possible to satisfy the assignment examples of each DM using individual weights and majority threshold parameters without using veto thresholds.

ICLV, or Infer Category Limits with Vetoes, finds, if possible, a set of profiles such that it is possible to satisfy the assignment examples of each DM using individual weights and majority threshold parameters and using veto thresholds if necessary. This is a generalization of the first program but they are presented in order of increasing complexity.

CWR, or Compute Weights Restrictions, having fixed a set of shared profiles, computes a measure indicating the remaining latitude for each DM regarding their possibility of choosing the weights ordering on the criteria.

These inference programs may be used in a process aiming to build a consensus by progressively reaching a common preferential model. These tools should be combined with other decision aiding tools, therefore allowing to build a comprehensive decision aiding process.

This technical report is mainly based on the disaggregation programs proposed in that paper, which should be read before reading this report. (First part of this introduction is reproduced from that article.) The MPs have been implemented in J-MCDA, an open source software for Multicriteria Decision Aiding (MCDA). This document gives detailed technical informations about how the Electre Tri disaggregation programs proposed in the companion article have been implemented, including choice of constants and strategies to avoid numerical errors.

More generally, several articles have appeared proposing Linear Programs (LPs) and Mixed Integer Programs (MIPs) that aim to obtain Electre Tri like preference models on the basis of assignment examples from the Decision Makers (DMs). Most often, the articles do not include discussions concerning the problem of numerical imprecision, hence, implementing the proposed Mathematical Programs (MPs) correctly can be difficult. It is the opinion of the author of this document that the current situation where articles are often published with no sufficient technical details to enable easy implementation is not fully satisfactory. This hinders possibility to reproduce or exploit the published results. Determining relevant technical informations can be very time consuming, and in the current situation, it may be considered that the usual quality assessment process in the OR community (perhaps also in other scientific communities) do not sufficently value such work. More generally, it might be considered important to ease reproducibility of the published results. Suggestions to improve the situation include favoring development of open standards for data exchange, and mandating their use when applicable; favoring development of open source softwares.

1.1 Numerical errors and mathematical programs

Koch et al. [6] gives a good overview of problems related to numerical imprecision when solving mathematical programs. The following paragraph is reproduced from their section 3.1.

Most MIP solvers (...) are based on floating-point arithmetic and work with tolerances to check solutions for feasibility and to decide on optimality. In their feasibility tests, solvers typically consider absolute tolerances for the integrality constraints and relative ones for linear constraints. Some normalize the activity of linear constraints individually, others directly scale the constraint matrix. The tolerances affect solution times and solution accuracy, normally in opposite

ways (...). If one fixes all integer variables from the reported solution to the closest integer value and recomputes the continuous variables by solving the resulting LP with exact arithmetic, some of these post-processed solutions turn out to be infeasible with respect to exact arithmetic and zero tolerances. (...) [This means] that the computed solution lies outside the feasible area described by the input file, but inside the extended feasible area created by reading in the problem and introducing tolerances.

This document refers to such situation where the solver considers some values as satisfying some constraints when these values, in exact arithmetic, do not satisfy the constraints, as a situation of numerical error. In the problems we are concerned with, such numerical errors may have important consequences. The MPs considered in this document are designed to satisfy all assignment examples from the DMs. If no particular measures are taken against numerical errors, a solver may report a solution which it considers feasible but which, when applied again to the original problem, does not satisfy all assignment examples.

In order to avoid such numerical errors, this document describes how some constraints must be modified by adding appropriate tolerance values. It also describes how the constants the MPs use should be chosen to make sure their use do not artificially reduce the set of feasible solutions. Finally, it includes some simple extentions and optimizations reducing the size of the resulting MP that have not been included in the companion article for simplicity. Tags have been added to constraints following the names used in the source code and displayed in files that can be exported from the library.

1.2 Constants used in the programs

- $\underline{g_j}$ and $\overline{g_j}$, the worst and best performances on the criterion j. Defining $\underline{g_j}$ lower than the worst performance on the criterion j is also acceptable, it must only represent a lower bound on the performances reached by any alternative on that criterion. Similar remark holds for $\overline{g_j}$.

- $\forall j \in J : \delta_j$, a value smaller than the minimal difference between any two different performances on j ($\forall j \in J : min_j = \min_{a,a' \in A} |g_j(a) - g_j(a')|$) divided by k. This leaves space to fit $k-1$ profiles into the interval $[g_j(a), g_j(a')]$ corresponding to any two different alternatives performances on j and separating each profile by δ_j. If there are no two different performances for a given criterion, δ_j is defined arbitrarily to 1. It is used to transform strict inequality constraints into loose constraints in Constraints (b, increase), (C, floor), (C, ceiling), (V, floor), (V, ceiling), (V, used). We used $min_j/(k+1)$.

- $\forall j \in J : \delta_j^b$, the "profile margin", maximal separation between the extreme profiles (resp. best and worst) and the extreme performances (resp. $\overline{g_j}$ and $\underline{g_j}$). We want to let the successive profiles be at least δ_j apart from each other, and at least δ_j apart from the extreme performances, thus, δ_j^b must be defined so that $\delta_j^b \geq (k-1)\delta_j$, in order to let all profiles be greater than the best performance, or lower than the worst performance, if necessary. We used $\delta_j^b = k\delta_j$. It is used to define the bounds of the profiles b_1 and b_{k-1}, and indirectly in constraints involving M_j.

- $\forall j \in J : M_j = \overline{g_j} - \underline{g_j} + \delta_j^b$, a scaling factor representing the maximal performance difference on the criterion j including one extreme profile. It is used in Constraints (C, floor), (C, ceiling), (V, floor), (V, ceiling), (V, used).

- δ_λ, an arbitrary small positive value, used in Constraints (support, ceiling) and (support, ceiling-v). We used $\delta_\lambda = 0.001$.

- δ_j^v, a value used in the definition of the veto variables ranges and in Constraints (V, ceiling), which must be greater than δ_j. We used $\delta_j * 2$.

2 Infer Profiles (IP)

2.1 Stating the problem

Having a set of alternatives A, categories $c_1, \ldots, c_h, \ldots, c_k$ ordered by preference where c_1 is the least preferred one, a set of profiles $B = \{b_1, \ldots, b_{k-1}\}$, where b_h is the upper profile to category c_h and lower profile to category c_{h+1}, a set of criteria indices J, the evaluations of the alternatives $g_j(a), \forall a \in A, j \in J$, a set of DMs $\mathcal{L}$, assignment examples E^l, the goal of IP is to determine the performances of profiles $g_j(b_h), \forall j \in J, 1 \leq h \leq k-1$ shared among the DMs, together with individual weights w_j^l and majority thresholds λ^l, matching all assignment examples. For each DM $l \in \mathcal{L}$, the set of examples E^l is the set of pairs $(a, b_h) \in A \times B$ specifying that the alternative a is assigned to the category c_h by l. We write $(a \underset{l}{\rightarrow} h)$ to denote such an assignment example.

Our programs are designed to be correctly defined if $k = 1$. It is useful to have a working program working even in such a trivial case for testing purposes and to make it as general as possible.

Note that the lowest profile, b_0, must be outranked by every alternative (because they must go into at least category c_1, the worst one) and hence the performances $g_j(b_0)$ on every criterion necessarily consist in performances lower than every other performances on the same criterion. The upper profile of the best category, that would be b_k, is used only when an other assignment procedure known as the optimistic procedure is used. Therefore, the lowest and best profiles are not considered in our inference programs. Satisfying these assignment examples can be written as follows:

$$\forall l \in \mathcal{L} : \forall (a \underset{l}{\rightarrow} h) \in E^l : \sum_{j \in J : g_j(a) \geq g_j(b_{h-1})} w_j^l \geq \lambda^l, \text{ and} \tag{1}$$

$$\forall l \in \mathcal{L} : \forall (a \underset{l}{\rightarrow} h) \in E^l : \sum_{j \in J : g_j(a) \geq g_j(b_h)} w_j^l < \lambda^l. \tag{2}$$

Equation (1) ensures that the example alternative is assigned to a category at least as good as c_h, and (2) makes sure that it is assigned to a category not better than c_h.

Hereafter we present a mathematical program finding adequate profiles without using veto thresholds. The case where veto thresholds are allowed is considered later in Section 3.

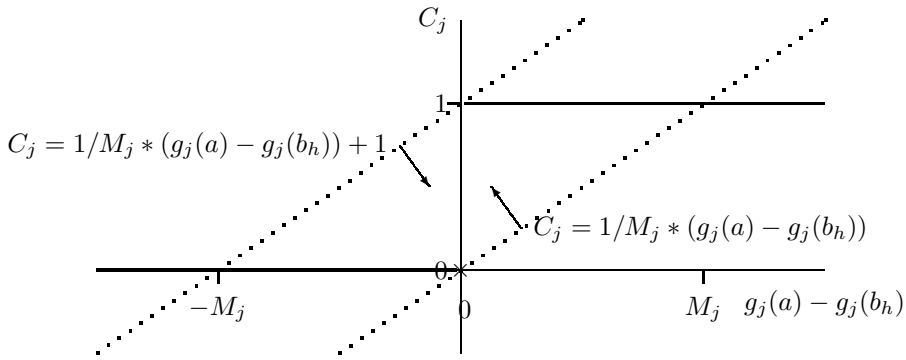

Figure 1: constraining C_j to the appropriate value

2.2 Constraints

$\forall j \in J, a \in A, 1 \leq h \leq k-1$:

$$\frac{1}{M_j}(g_j(a) - g_j(b_h)) < C_j(a, b_h), \tag{3}$$

$$C_j(a, b_h) \leq \frac{1}{M_j}(g_j(a) - g_j(b_h)) + 1. \tag{4}$$

Constraints (3) and (4) define $C_j(a, b_h)$, binary variables indicating the agreement of the criterion j to say that a is at least as good as b_h. See Figure 1. In order to transform (3) into a non strict inequality, δ_j may be used. It is defined so that adding this value to the constraint does not restrict the set of acceptable values for $g_j(b_h)$. It is used to change the strict inequality to a loose inequality: $\forall j \in J, a \in A, 1 \leq h \leq k-1$,

$$\frac{1}{M_j + \delta_j}(g_j(a) - g_j(b_h) + \delta_j) \leq C_j(a, b_h). \tag{C, floor}$$

Constraints (4) are exposed to numerical errors. The solver could find a solution with $g_j(b_h) = g_j(a) + \varepsilon$, for some alternative a, criterion j, profile b_h, with ε a small positive value, and consider $C_j(a, b_h) = 1$ although $C_j(a, b_h)$ should equal zero as the profile value is higher than the alternative performance. This is so because it could happen that $C_j(a, b_h) = 1$ satisfies the constraint (4) stating that $C_j(a, b_h) \leq \frac{-\varepsilon}{M_j} + 1$ because of the solver tolerance. To avoid this situation, Constraints (4) should be transformed to

$$C_j(a, b_h) \leq \frac{1}{M_j + x_j}(g_j(a) - g_j(b_h) - x_j) + 1. \tag{C, ceiling}$$

Having t the tolerance for integer values used by the solver, $\frac{x_j}{M_j + x_j}$ should be at least t, thus $x_j \geq (M_j + x_j)t$. This makes the constraint stronger, as it now holds that $g_j(a) < g_j(b_h) + x_j \Rightarrow C_j(a, b_h) = 0$. The new constraint form prevents situations where $g_j(b_h) < g_j(a) < g_j(b_h) + x_j$, thus, prevents to choose the performance of the profile such that $\exists a \in A \mid g_j(a) - x_j < g_j(b_h) < g_j(a)$. In

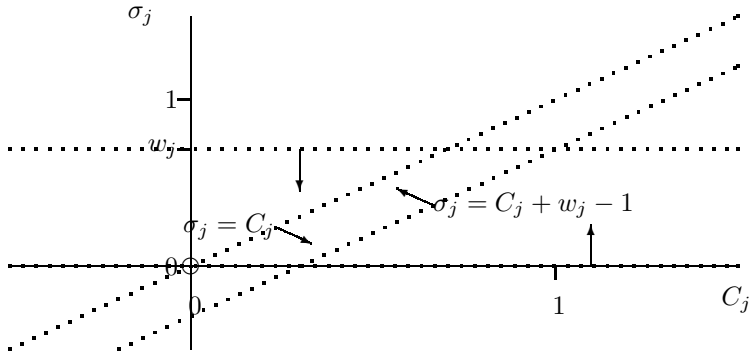

Figure 2: constraining σ_j^l to the appropriate value

order to avoid reducing the feasible solution space, x_j has to be small enough. A practical solution is to take $x_j = \delta_j$. For this solution to be implementable, it must hold that $\frac{\delta_j}{M_j+\delta_j} \geq t, \forall j \in J$. For example, the CPLEX solver version 12.3 uses by default $t = \frac{1}{10^5}$. This implies that the proposed strategy does not apply if the proportion of the minimal difference over the maximal difference of performance is too small. A solution to this problem is to rescale the data. An alternative strategy is to avoid using the values given by the solver for $g_j(b_h)$, instead of changing constraint (4). The profiles evaluations may instead be deduced from the solver values for $C_j(a, b_h)$. This may be a better strategy as it imposes no restriction on the input data, but it requires more efforts to implement.

The following constraints are used to define the variables $\sigma_j^l(a, b_h)$. These variables are defined as non negative. $\forall l \in \mathcal{L}, j \in J, a \in A^*, 1 \leq h \leq k-1$:

$$\begin{cases} \sigma_j^l(a, b_h) \leq C_j(a, b_h) & (\sigma, \text{ ceiling } C) \\ \sigma_j^l(a, b_h) \leq w_j^l & (\sigma, \text{ ceiling } w) \\ C_j(a, b_h) - 1 + w_j^l \leq \sigma_j^l(a, b_h). & (\sigma, \text{ bottom}) \end{cases}$$

These variables represent the sum of the support for saying that a is at least as good as b_h. The constraints ensure that $\sigma_j^l(a, b_h) = w_j^l C_j(a, b_h)$ while avoiding the use of a non-linear expression. See also Figure 2.

Then we need to ensure that the sum of supports for each examples are lower and greater than the related majority thresholds (Constraints (5), (6)).

$$\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} h) \in E^l, h > 1 : \sum_{j \in J} \sigma_j^l(a, b_{h-1}) \geq \lambda^l. \tag{5}$$

$$\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} h) \in E^l, h \leq k-1 : \sum_{j \in J} \sigma_j^l(a, b_h) \leq \lambda^l - \delta_\lambda, \tag{6}$$

Constraints (5) are subject to numerical errors. It is possible (and it has been observed with CPLEX) that $\sum_{j \in J} \sigma_j^l(a, b_{h-1}) = \lambda^l - \varepsilon$, for a given a, h, l, with a small positive ε value, such that the solver considers the constraint is satisfied when it is strictly speaking not. Modifying the constraint by adding

a small positive δ'_λ is not a good solution because it would restrict the feasible space, in particular it would exclude solutions where $\lambda = 0.5$ involving two disjoint winning coalitions. Moreover, this problem requires a more general solution. Even when no MP is involved, imprecision may be introduced when reading the weights values from a file, or obtaining them from an other source, or when computing the sum itself, so that the computed sum of weights may well be slightly below the λ value even though the true weight values may be greater than or equal to λ. The solution we adopted for this general problem is the following. When comparing the sum of weight values to the majority threshold λ, we use a small tolerance margin ε (configurable in our library). If $\sum_{j \in J | g_j(a) \geq g_j(b_h)} w_j \geq \lambda^l - \varepsilon$, we consider that the alternative outranks the profile, as if $\sum_{j \in J | g_j(a) \geq g_j(b_h)} w_j(a, b_{h-1}) \geq \lambda^l$ held. This requires to make sure that the true weight values (considering no numerical imprecision) are such that no coalitions have a value in $[\lambda - \varepsilon, \lambda[$, otherwize the tolerance margin modifies the intended result. In the case of the MP, suffices to take $\varepsilon < \delta_\lambda$ (we suggest $\varepsilon = \delta_\lambda/2$), because Constraints (6) make sure that either the coalition is less than or equal to $\lambda^l - \delta_\lambda$, or it should be considered as a winning coalition. This solves the MP imprecision problem as long as $\varepsilon > t$, with t the imprecision tolerance used by the solver for linear constraints (CPLEX version 12.3 uses $t = \frac{1}{10^6}$ by default). One problem remains when using this strategy. The weights solution values taken from the solver solution to the MP have to be interpreted using an appropriate ε value, that the user has to know. A solution to that problem would be to recompute equivalent weight and λ values (not posing any numerical imprecision risk), that have the same winning coalitions, and show the recomputed values to the user instead of the direct values from the solver.

Constraint (weights, sum) ensure that the weights sum to one. $\forall l \in \mathcal{L}$:

$$\sum_{j \in J} w_j^l = 1. \qquad \text{(weights, sum)}$$

This is obviously subject to numerical imprecision, but it is very unlikely to cause any practical problem.

Constraints (b, increase) make sure that the profiles are ordered correctly. $\forall j \in J, 2 \leq h \leq k-1$:

$$g_j(b_{h-1}) \leq g_j(b_h) - \delta_j. \qquad (b\text{, increase})$$

Ensuring that the profiles are separated by δ_j permits to avoid numerical errors where a profile would be higher than a worst one by a very small margin such as 1e-10. Separating them by δ_j does not restrict the possibility that they are semantically equal, as that value is small enough. Observe that it is therefore possible that two profiles have semantically equal performance on all criteria, thus are exactly the same. A necessary condition for this to happen is that at least one category has no assignment examples.

Finally, note that it is necessary to define the bounds of the profiles (if $k \geq 2$), to be able to use the scaling constants in the rest of the program. The bounds must be defined only for the worst and best profiles as Constraints (b, increase) ensure that the other profiles also satisfy the bounds. Bounds are defined as follows, $\forall j \in J$:

$$\begin{cases} \underline{g_j} - \delta_j^b \leq g_j(b_1), \\ \qquad\qquad g_j(b_{k-1}) \leq \overline{g_j} + \delta_j^b. \end{cases}$$

2.3 Using intervals instead of crisp assignments

The user may wish to provide assignment examples in the form of a contiguous interval of categories where an alternative may be assigned to, instead of a single category. In the case of continuous intervals, the mathematical program is easily transformed. If the assignments were allowed to be non contiguous, e.g. alternative a may go to category 1 *or* 3, the change would be more complex.

Now E^l contains, for each example involving an alternative a, two pairs (a, h) with the lowest and highest category the alternative may be assigned to. Let us write $(a \underset{l}{\rightarrow} \underline{h})$ and $(a \underset{l}{\rightarrow} \overline{h})$ the lower category and the upper category the alternative a should be assigned to according to the decision maker l. Note that $(a \underset{l}{\rightarrow} \underline{h}) = (a \underset{l}{\rightarrow} \overline{h})$ is possible, if the alternative is to be assigned into a single category. This change requires to modify Constraints (5) and (6).

$\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} \underline{h}) \in E^l, \underline{h} \geq 2$:

$$\sum_{j \in J} \sigma_j^l(a, b_{\underline{h}-1}) \geq \lambda^l. \qquad \text{(support, floor)}$$

$\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} \overline{h}) \in E^l, \overline{h} \leq k-1$:

$$\sum_{j \in J} \sigma_j^l(a, b_{\overline{h}}) \leq \lambda^l - \delta_\lambda. \qquad \text{(support, ceiling)}$$

2.4 Objective function

If we want to maximize the separation between the sum of support and the majority thresholds, we may maximize a slack variable s as objective of the MIP and replace Constraints (support, floor) and (support, ceiling) with the following ones. $\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} \underline{h}) \in E^l, \underline{h} \geq 2$:

$$\sum_{j \in J} \sigma_j^l(a, b_{\underline{h}-1}) \geq \lambda^l + s,$$

and $\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} \overline{h}) \in E^l, \overline{h} \leq k-1$:

$$\sum_{j \in J} \sigma_j^l(a, b_{\overline{h}}) + s \leq \lambda^l - \delta_\lambda.$$

2.5 Reduced mathematical program

The following program uses less constraints than what has been described previously by using the fact that only the technical variables required to ensure satisfaction of the assignment examples are necessary. It is thus possible to reduce the size of the program. For example, it is not necessary to define $C_j(a, b_h)$ variables for each alternative a and profile b_h. Let us note P^l the set of pairs of alternatives and profiles that we have to consider to satisfy assignment examples of DM l. For each assignment example $(a \underset{l}{\rightarrow} \underline{h})$, if $h \geq 2$, we have to consider the comparison of a to b_{h-1} to make sure alternative a is at least as good as

the profile and can reach category h. Similarily, for each assignment example $(a \underset{l}{\rightarrow} \overline{h})$, if $h \leq k-1$, the pair (a, b_h) must be considered. Finally, we have:

$$\begin{aligned}
\underline{P^l} &= \left\{ (a, b_{h-1}), \forall (a \underset{l}{\rightarrow} \underline{h}) \in E^l, \underline{h} \geq 2 \right\}, \\
\overline{P^l} &= \left\{ (a, b_h), \forall (a \underset{l}{\rightarrow} \overline{h}) \in E^l, \overline{h} \leq k-1 \right\}, \\
P^l &= \underline{P^l} \cup \overline{P^l}.
\end{aligned}$$

Having a set of alternatives A, a set of criteria indices J, the evaluations of the alternatives $g_j(a), \forall a \in A, j \in J$, the number of categories k, a set of DMs $\mathcal{L}$, assignment examples E^l, determine the performances of the profiles $g_j(b_h), \forall j \in J, 1 \leq h \leq k-1$, together with individual weights w_j^l and majority thresholds λ^l, satisfying the following constraints.

$\forall l \in \mathcal{L}$:

$$\sum_{j \in J} w_j^l = 1. \qquad \text{(weights, sum)}$$

$\forall j \in J, 2 \leq h \leq k-1$:

$$g_j(b_{h-1}) \leq g_j(b_h) - \delta_j. \qquad (b\text{, increase})$$

$\forall l \in \mathcal{L}, (a, b_h) \in P^l, j \in J$:

$$\begin{cases}
\dfrac{1}{M_j + \delta_j}(g_j(a) - g_j(b_h) + \delta_j) \leq C_j(a, b_h), & (C\text{, floor}) \\
C_j(a, b_h) \leq \dfrac{1}{M_j + \delta_j}(g_j(a) - g_j(b_h) - \delta_j) + 1. & (C\text{, ceiling})
\end{cases}$$

$\forall l \in \mathcal{L}, (a, b_h) \in P^l, j \in J$:

$$\begin{cases}
C_j(a, b_h) + w_j^l - 1 \leq \sigma_j^l(a, b_h), & (\sigma\text{, floor}) \\
\sigma_j^l(a, b_h) \leq C_j(a, b_h), & (\sigma\text{, ceiling } C) \\
\sigma_j^l(a, b_h) \leq w_j^l. & (\sigma\text{, ceiling } w)
\end{cases}$$

$\forall l \in \mathcal{L}, (a, b_h) \in \underline{P^l}$:

$$\sum_{j \in J} \sigma_j^l(a, b_h) \geq \lambda^l. \qquad \text{(support, floor)}$$

$\forall l \in \mathcal{L}, (a, b_h) \in \overline{P^l}$:

$$\sum_{j \in J} \sigma_j^l(a, b_h) \leq \lambda^l - \delta_\lambda. \qquad \text{(support, ceiling)}$$

The constants $\underline{g_j}$, $\overline{g_j}$, M_j, δ_j, δ_j^b, and δ_λ are defined in Section 1.2. The variables $C_j(a, b_h)$ are binaries, $g_j(b_h)$ are real, with $\underline{g_j} - \delta_j^b \leq g_j(b_1)$ and $g_j(b_{k-1}) \leq \overline{g_j} + \delta_j^b$, w_j^l and $\sigma_j^l(a, b_h)$ are real and non negative, λ^l are in $[0.5, 1]$.

3 Infer Profiles with Vetoes (IPV)

Suppose now that the DMs are ready to accept veto thresholds in the individual preference models, i.e. when searching for common profiles, it is deemed acceptable to use shared veto thresholds to satisfy the individual assignment examples. This can enable to find common profiles in situations where it would not be possible to satisfy all examples without vetoes.

This program can be seen as a generalization of the previous one, except for the change in the objective function, as it tries to find solutions having zero vetoes. However we choose to present the programs in an increasing order of complexity. For simplicity, we also consider that the DMs share the vetoes. The same approach would be applicable for searching individual vetoes, with an objective function that could e.g. minimize the number of individual vetoes used or minimize the number of DMs using some vetoes. Sharing the vetoes also reduces the number of binary variables and reduces the risk that the resulting preference model would overfit the provided data.

The mathematical program is based on the previous one, with a few additions and changes. The veto situations are modeled as follows: a veto threshold v_j^h (a variable in our problem) is associated with each of the criteria $j \in J$ and profile index $1 \leq h \leq k-1$. We also need binary variables $V_j(a, b_h), \forall j \in J, a \in A, 1 \leq h \leq k-1$, equal to one iff there is a veto situation between a and b_h. When for any criterion and profile the evaluation $g_j(a)$ is strictly lower than the corresponding v_j^h veto threshold, the alternative may not outrank the profile.

Not using a veto threshold is equivalent to setting $v_j^h < \underline{g_j}$, the worst performances on the criterion j. We thus define the variables v_j^h as having a range $\underline{g_j} - \delta_j^v \leq v_j^h \leq g_j(b_h)$, where $\delta_j^v > \delta_j$.

The implementation uses a more classical definition of the veto. E.g. Roy [9] defines a veto threshold $r_j \geq p_j$, with $p_j > 0$ the preference threshold, which prevents a from outranking b iff $g_j(a) < g_j(b) - r_j$. In our implementation $\boldsymbol{v_j^h} = g_j(b_h) - v_j^h$ (or: $v_j^h = g_j(b_h) - \boldsymbol{v_j^h}$). An alternative a may not go into category h if $g_j(b_h) - g_j(a) > \boldsymbol{v_j^h}$, which is equivalent to $g_j(a) < v_j^h$. Thus, not using a veto threshold is equivalent to setting $\boldsymbol{v_j^h} \geq g_j(b_h) - \underline{g_j}$, and the variables $\boldsymbol{v_j^h}$ must have a range defined as at least $0 \leq \boldsymbol{v_j^h} \leq g_j(b_h) - \underline{g_j} + \delta_j^v$, with $\delta_j^v > \delta_j$. We choose to define all upper bound for variables $\boldsymbol{v_j^h}$ as $\boldsymbol{v_j^h} \leq M_j + \delta_j^v$, for simplicity. Using a classical definition of the veto makes it more efficient to implement the case of constant veto thresholds, where the value of $\boldsymbol{v_j^h}$ does not depend on h.

In order to clarify the relation between the implementation and the companion article, the following subsection gives both form of constraints, those with the companion article definition of the veto v_j^h and those with the implementation definition of the veto $\boldsymbol{v_j^h}$.

3.1 Additional constraints

$\forall j \in J, 2 \leq h \leq k-1$:

$$v_j^{h-1} \leq v_j^h$$

ensures that the veto thresholds are correctly ordered. Equivalently:

$$\boldsymbol{v_j^h} - \boldsymbol{v_j^{h-1}} \leq g_j(b_h) - g_j(b_{h-1}). \qquad (v, \text{order})$$

The following constraints define the binary variables $V_j(a, b_h)$ indicating the existence of a veto involving alternative a and category h. $\forall j \in J, a \in A, 1 \leq h \leq k-1$:

$$\frac{v_j^h - g_j(a) + \delta_j}{M_j + \delta_j} \leq V_j(a, b_h) \leq \frac{v_j^h - g_j(a)}{2M_j + \delta_j^v} + 1,$$

or equivalently:

$$\frac{g_j(b_h) - g_j(a) - \boldsymbol{v_j^h} + \delta_j}{M_j + \delta_j} \leq V_j(a, b_h), \qquad (V, \text{ floor})$$

$$V_j(a, b_h) \leq \frac{g_j(b_h) - g_j(a) - \boldsymbol{v_j^h} - \delta_j}{2M_j + \delta_j^v + \delta_j} + 1. \qquad (V, \text{ ceiling})$$

Note that $g_j(b_h) - g_j(a) - \boldsymbol{v_j^h} \in [-2M_j - \delta_j^v, M_j]$. Substracting δ_j in (V, ceiling) permits to avoid numerical errors: (V, floor) and (V, ceiling) together forbid that there exists $g_j(b_h) - g_j(a) = \boldsymbol{v_j^h}$, therefore allowing a security margin for numerical imprecision.

$\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} \underline{h}) \in E^l, \underline{h} \geq 2$:

$$\sum_{j \in J} \sigma_j^l(a, b_{\underline{h}-1}) \geq \lambda^l + \sum_{j \in J} V_j(a, b_{\underline{h}-1}). \qquad \text{(support, floor-v)}$$

$\forall l \in \mathcal{L}, \forall (a \underset{l}{\rightarrow} \overline{h}) \in E^l, \overline{h} \leq k-1$:

$$\sum_{j \in J} \sigma_j^l(a, b_{\overline{h}}) \leq \lambda^l - \delta_\lambda + \sum_{j \in J} V_j(a, b_{\overline{h}}). \qquad \text{(support, ceiling-v)}$$

Constraints (support, floor-v), (support, ceiling-v) take the veto into account when computing the assignments. $\sum_{j \in J} V_j(a, b_h)$ accounts for the existence of a veto situation between a and b_h.

To minimize the number of vetoes used, we need a binary variable for each criterion, V_j, capturing the fact that a veto is used for this criterion. The following sufficient condition for considering that a veto is not used holds, $\forall j \in J, 1 \leq h \leq k-1$:

$$v_j^h < \underline{g_j} \Rightarrow V_j = 0.$$

Thanks to the ordering of the vetoes (Constraints (v, order)), it can be refined as it must hold only for $h = k-1$. In the linear model we only need to constrain V_j to one if necessary, thus having defined V_j as a binary variable, $\forall j \in J$:

$$V_j \geq \frac{v_j^{k-1} - \underline{g_j} + \delta_j}{M_j + \delta_j},$$

or equivalently:

$$V_j \geq \frac{g_j(b_{k-1}) - \underline{g_j} + \delta_j - \boldsymbol{v_j^{k-1}}}{M_j + \delta_j}. \qquad (V, \text{ used})$$

This formulation is much more economical than the formulation used in the companion paper ($V_j \geq V_j(a, b_h), \forall a, h$), but requires more justification. Let us say that a veto, associated to a profile b_h and criterion j, is possibly used iff $\boldsymbol{v_j^h} \geq g_j(b_h) - \underline{g_j}$. By this phrase we mean that the veto is defined to a value which,

considering $\underline{g_j}$ is the lowest possible value on the scale of performances of j, makes it possibly intervene to veto an outranking situation. If the performance $\underline{g_j}$ is reached by some alternative in the set of alternatives used as example, then the veto value v_j^h is reached by that alternative. If the veto value v_j^h is reached by at least one alternative, we say that the veto is used. Note that is an abuse of language, because it might be that the veto is not necessary for the outranking situation not to hold. If the veto is necessary to prevent the outranking situation for at least one alternative in the example alternative set, that is, if the concordance is greater than or equal to the majority threshold for the relevant alternative and profile, we say that the veto is useful. Recall we authorize $\underline{g_j}$ to be lower than all alternatives evaluations.

It follows from Constraints (V, used) that if a veto is possibly used, then $V_j = 1$ and if $V_j = 0$, then no veto is possibly used, thus, no veto is ever used, for criterion j. The V_j variables should be defined if and only if the objective function includes minimizing the number of vetoes. Thanks to that objective, $V_j = 1$ requires that the veto on criterion j is useful. We also have that useful implies possibly used, thus it holds that $V_j = 1$ iff the veto on criterion j is useful.

Should the user want to search for preference models where some criteria are forbidden to use a veto, it is possible to force non use of a veto for a given criterion j by constraining the value of $\boldsymbol{v_j^{k-1}}$ to be at least $g_j(b_{k-1}) - \underline{g_j}$, or $\boldsymbol{v_j^{k-1}} \geq M_j$, or by simply not defining relevant $\boldsymbol{v_j^{k-1}}$ and V_j variables. Note also that it is meaningless to force V_j to one: supplementary constraints would be needed to make sure the veto is used. In the implementation, it is also possible to constrain a veto threshold to some value. This is implemented by simply restricting the range of the appropriate $\boldsymbol{v_j^h}$ variable. It does not guarantee that the veto is indeed useful. Remark that forcing the value of $\boldsymbol{v_j^h}$ would be a problem if the choice of $g_j(b_h)$ would lead to $g_j(b_h) - \underline{g_j} \leq \boldsymbol{v_j^h} < g_j(b_h) - \underline{g_j} + \delta_j^v$. In that case we would have $V_j = 1$ although the veto would be unused. However, this situation will not happen because the objective is to minimize the sum of the V_j variables, therefore leading the solver to choose a value for $g_j(b_h)$ which does not cause this situation. This also shows why these constraints should be used only if the objective involves minimizing the sum of the V_j variables.

3.2 Nul weights and veto thresholds

Procedures based on Electre Tri take different decisions, depending on the variant used, when a veto is bound to a criterion j having a nul weight. An ambiguous situation happens when the concordance, thus the sum of weights of the criteria in favor of an alternative a being at least as good as a profile b_h, equals one, and criterion j has a veto against that affirmation. In the original Electre Tri definition [9], a concordance of one is a sufficient condition to determine outranking, which opposes the rule that the veto should make outranking impossible. If a veto can be defined on a nul weight criterion, such an ambiguous condition may happen. Let us qualify a solution involving at least one criterion where a veto is defined and having a nul weight as a solution with zero-weight vetoes. In the recently axiomatized version of Electre Tri [1, 2], it is admitted that a veto is defined on a criterion which has no role in any winning coalition. In such a case, the veto plays its role, forbidding the outranking result. In our

implementation, the question arises whether such a situation should be allowed. We decide that if a veto is defined on a zero weight threshold, it should play its role and veto the outranking, as suggested by Bouyssou and Marchant. With such a decision, let us show that by forbidding zero-weight vetoes, we do not reduce expressiveness of the model, or conversely, adding such a possibility does not augment the expressiveness of the model. We must show that if no solution exists without zero-weights vetoes, then no solution exists allowing zero-weight vetoes. Conversely, let us show that if a solution exists involving zero-weight vetoes, then a solution exists without zero-weight vetoes.

Consider an Electre Tri preference model M involving zero-weight vetoes, let us transform it to M' which does not involve zero-weight vetoes and is equivalent, thus sorts all possible alternatives to the same categories. Let us note j a criterion with zero weight and a veto in M. We show how to transform the solution to an equivalent one in which j has a non zero weight. If several criteria have zero weight and a veto, the procedure may be repeated.

Let us note the vetoes and profile levels in M bound to criterion j as $\boldsymbol{v_j^h}$ and $g_j(b_h), \forall h$. As criterion j has zero weight, the profiles may be transformed to any values $g'_j(b_h)$, for all h, keeping the resulting model equivalent to the original one. We may also choose new veto values $\boldsymbol{v'^h_j}$ but the vetoes must operate in the same conditions, otherwise the transformed solution is not equivalent to the one found, thus we must have $g'_j(b_h) - \boldsymbol{v'^h_j}$ close enough to $g_j(b_h) - \boldsymbol{v_j^h}$. Define $g'_j(b_h) = g_j(b_h) - \boldsymbol{v_j^h}$ and $\boldsymbol{v'^h_j} = 0$, so that the veto occurs for a category h and alternative a exactly when the alternative is less good than the corresponding profile according to j. It is then possible to associate a non nul weight to the criterion j. When the criterion is not in favor of $a > b$, its weight does not matter as the veto will be decisive. In all cases where its weight matter, it will be part of the winning coalition. We may thus give this criterion any weight (e.g., 0.1), and raise the majority threshold by the same value.

The above result proves that forbidding vetoes on zero-weight criteria does not reduce the set of feasible solutions when a single preference model is involved. Observe however that this holds in the case of a single DM. In our case of multiple DMs sharing profiles, adding such constraints could reduce the set of feasible solution. We therefore chose to authorize solutions involving zero-weight vetoes.

3.3 Objective function

In most situations it is probably reasonable to find a solution involving the least possible number of vetoes: allowing too many veto thresholds to be used may lead to an over-fitting of the model with ad-hoc veto thresholds. The objective function should then be to minimize the sum of the V_j variables:

$$min \sum_{j \in J} V_j.$$

An alternative could be e.g. to minimize the number of situations where a veto is used.

3.4 Reduced mathematical program

Having a set of alternatives A, a set of criteria indices J, the evaluations of the alternatives $g_j(a), \forall a \in A, j \in J$, the number of categories k, a set of DMs $\mathcal{L}$, assignment examples E^l, determine the performances of the profiles $g_j(b_h)$ and veto levels $\boldsymbol{v_j^h}$, $\forall j \in J, 1 \leq h \leq k-1$, together with individual weights w_j^l and majority thresholds λ^l, minimizing $\sum_{j \in J} V_j$ subject to the following constraints. Note that this MP is also smaller than the one published in the companion article.

$\forall l \in \mathcal{L}$:

$$\sum_{j \in J} w_j^l = 1. \qquad \text{(weights, sum)}$$

$\forall j \in J, 2 \leq h \leq k-1$:

$$g_j(b_{h-1}) \leq g_j(b_h) - \delta_j. \qquad (b\text{, increase})$$

$\forall l \in \mathcal{L}, (a, b_h) \in P^l, j \in J$:

$$\begin{cases} \dfrac{1}{M_j + \delta_j}(g_j(a) - g_j(b_h) + \delta_j) \leq C_j(a, b_h), & (C\text{, floor}) \\ C_j(a, b_h) \leq \dfrac{1}{M_j + \delta_j}(g_j(a) - g_j(b_h) - \delta_j) + 1. & (C\text{, ceiling}) \end{cases}$$

$\forall l \in \mathcal{L}, (a, b_h) \in P^l, j \in J$:

$$\begin{cases} C_j(a, b_h) + w_j^l - 1 \leq \sigma_j^l(a, b_h), & (\sigma\text{, floor}) \\ \sigma_j^l(a, b_h) \leq C_j(a, b_h), & (\sigma\text{, ceiling } C) \\ \sigma_j^l(a, b_h) \leq w_j^l. & (\sigma\text{, ceiling } w) \end{cases}$$

$\forall j \in J, 2 \leq h \leq k-1$:

$$\boldsymbol{v_j^h} - \boldsymbol{v_j^{h-1}} \leq g_j(b_h) - g_j(b_{h-1}). \qquad (v\text{, order})$$

$\forall l \in \mathcal{L}, (a, b_h) \in P^l, j \in J$:

$$\begin{cases} \dfrac{g_j(b_h) - g_j(a) - \boldsymbol{v_j^h} + \delta_j}{M_j + \delta_j} \leq V_j(a, b_h), & (V\text{, floor}) \\ V_j(a, b_h) \leq \dfrac{g_j(b_h) - g_j(a) - \boldsymbol{v_j^h} - \delta_j}{2M_j + \delta_j^v + \delta_j} + 1. & (V\text{, ceiling}) \end{cases}$$

$\forall l \in \mathcal{L}, (a, b_h) \in \underline{P^l}$:

$$\sum_{j \in J} \sigma_j^l(a, b_h) \geq \lambda^l + \sum_{j \in J} V_j(a, b_h). \qquad \text{(support, floor-v)}$$

$\forall l \in \mathcal{L}, (a, b_h) \in \overline{P^l}$:

$$\sum_{j \in J} \sigma_j^l(a, b_h) \leq \lambda^l - \delta_\lambda + \sum_{j \in J} V_j(a, b_h). \qquad \text{(support, ceiling-v)}$$

$\forall j \in J$:

$$V_j \geq \frac{g_j(b_{k-1}) - \underline{g_j} + \delta_j - \boldsymbol{v_j^{k-1}}}{M_j + \delta_j}. \qquad (V, \text{ used})$$

The constants $\underline{g_j}$, $\overline{g_j}$, M_j, δ_j, δ_j^b, δ_j^v and δ_λ are defined in Section 1.2. The variables $C_j(a, b_h)$, $V_j(a, b_h)$ and V_j are binaries, $g_j(b_h)$ are real, with $\underline{g_j} - \delta_j^b \leq g_j(b_1)$ and $g_j(b_{k-1}) \leq \overline{g_j} + \delta_j^b$, w_j^l, $\sigma_j^l(a, b_h)$ are real and non negative, $\overline{\lambda}^l$ are in $[0.5, 1]$, $\boldsymbol{v_j^h}$ are in $[0, M_j + \delta_j^v]$.

4 Conclusion

We have presented in this techical report implementation details permitting to formulate several disaggregation programs as MPs while taking into account numerical imprecision thresholds used by solvers. The report also presents reduced MPs that permit a more efficient implementation than the one which was previously published. These algorithms are available in J-MCDA, a free (libre) open source software library. The report uses, as much as possible, the same technical terms and notations as are used in the source code and it may therefore also be useful as a detailed code documentation.

As is mentioned in the companion paper, some instance sizes are not currently solvable in a reasonable time using this approach. For example, instances involving more than eight criteria remain difficult to solve. One possibility to reduce the resolution time is to design resolution algorithms making use of the specific structure of the problems to solve. This is left for future work.

References

[1] Bouyssou, D. and Marchant, T. (2007a). An axiomatic approach to noncompensatory sorting methods in MCDM, I: The case of two categories. *European Journal of Operational Research*, 178(1):217–245.

[2] Bouyssou, D. and Marchant, T. (2007b). An axiomatic approach to noncompensatory sorting methods in MCDM, II: more than two categories. *European Journal of Operational Research*, 178(1):246–276.

[3] Cailloux, O., Meyer, P., and Mousseau, V. (2012). Eliciting ELECTRE TRI category limits for a group of decision makers. *European Journal of Operational Research*. Submitted.

[4] Figueira, J., Mousseau, V., and Roy, B. (2005). ELECTRE methods. In Figueira, J., Greco, S., and Ehrgott, M., editors, *Multiple Criteria Decision Analysis: State of the Art Surveys*, pages 133–162. Springer Verlag, Boston, Dordrecht, London. Chapter 4.

[5] Greco, S., Matarazzo, B., and Słowiński, R. (2002). Rough sets methodology for sorting problems in presence of multiple attributes and criteria. *European Journal of Operational Research*, 138(2):247–259.

[6] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D. E., and Wolter, K. (2011). MIPLIB 2010. *Mathematical Programming Computation*, 3(2):103–163.

[7] Mousseau, V., Słowiński, R., and Zielniewicz, P. (2000). A user-oriented implementation of the ELECTRE TRI method integrating preference elicitation support. *Computers & Operations Research*, 27(7-8):757–777.

[8] Perny, P. (1998). Multicriteria filtering methods based on Concordance/Non-Discordance principles. *Annals of Operations Research*, 80:137–167.

[9] Roy, B. (1991). The outranking approach and the foundations of ELECTRE methods. *Theory and Decision*, 31:49–73.