Decision-making systems: Implementing three ways to solve an Inv-NCS problem

Ariane Dalens, Lucas Sor, Magali Morin

February 1, 2022

Abstract

This project aims at presenting the concepts and methods for preference modeling and multi-criteria decision making. We will solve an Inv-NCS problem using three approaches: first with a Mixed-Integer Linear Program, then a SAT formulation and last a MaxSAT formulation.

Keywords: NCS, Inv-NCS, MR-Sort, MILP, SAT, MaxSAT

1 Introduction

Consider a situation in which a committee for a higher education program has to decide about the admission of students on the basis of their evaluations in 4 courses: mathematics (M), physics (P), literature (L) and history (H). Evaluations on all courses range in the $[\![0,20]\!]$ interval. To be accepted (A) in the program, the committee considers that a student should obtain at least 12 on a "majority" of courses, otherwise, the student is refused (R). From the committee point of view, all courses (criteria) do not have the same importance. To define the required majority of courses, the committee attaches a weight $w_j \geq 0$ to each course such that they sum to 1; a subset of courses $C \subseteq \{M, P, L, H\}$ is considered as a majority if $\sum_{j\in C} w_j \geq \lambda$, where $\lambda \in [0,1]$ is a required majority level.

You can access our gitlab to review our code.

Solving Inv-NCS problem is computationally difficult and have been proved to be NP-hard [7], to solve this issue we will implement 3 approaches: Mixed-Integer Linear programming [6], SAT [2] and Max-SAT [1].

To solve this problem we will implement:

1. an with a linear solver (gurobi) as a first approach
2. an Inv-NCS with a SAT solver to get better results
3. an Inv-NCS with a MaxSAT formulation to deal with noisy data

Our solutions will be evaluated based on:

1. computational time
2. ability to learn a data set and its generalization ability
3. adaptability to noisy data

2 Data set generation

2.1 MR-Sort generation

In order to generate a data set for the Inv-MR-Sort problem, our algorithm takes as arguments the following variables : The number of students and the numbers of grades they have. Then we randomly generate
- each student giving them random grades for each grade,

- the frontier vector containing the minimal grades to have to pass a given course,
- the weight vector and the threshold.

Using the generated data set of grades and the weights and threshold, we compute for each students whether or not they are accepted or refused.

2.2 NCS generation

As we have multiples classes, the algorithm needs the number of classes as an input to create the data set. We randomly generate students similarly to MR-Sort generation but now, as there are more classes, we have multiple frontiers. The weights vector and the threshold are also generated as in the MR-Sort algorithm. Now, to correctly classify students in each class, the algorithm looks at the highest class such that the student has a majority of grades above the class frontier.

2.3 Adding noise

In order to add noise, we added another input to our algorithm which is a number $p \in [0,1]$ such that for each student, they will be classified in a random class with probability p.

3 Theoretical explanation

3.1 Linear programming algorithm

In this section of the article we will discuss the MILP usage [5] [2] to solve the Inv-MR-Sort problem.

The problem statement is as follows :

We consider two sets of students A (accepted) and R (rejected) and that the coalitions of are represented using additive weights w_j and a majority threshold λ. That is to say that students are accepted if and only if the sum of the weight of their courses in which they obtain at least the minimum required mark is a majority (greater or equal to λ).

Our goal is to find the weight vector w and the threshold λ using linear programming and SAT solvers.

Let $b = (b_1, \ldots, b_n)$ the different minimal grades that one student must have to pass some course i.e. student s passes course $i \in [\![1,n]\!]$ if and only if his grade at course i noted s_i is such that $s_i \geq b_i$.

We can now define a boolean variable $\delta_i(s)$ such that $\delta_i(s) = 0$ if the student fails course i and $\delta_i(s) = 1$ if student passes course i. This can be summed up with the formula

$$\forall s, i \quad \delta_i(s) = 1 \Leftrightarrow s_i \geq b_i$$

In order to put this behavior in a linear programming algorithm we can proceed as follows. We consider an arbitrarily large number M and add the condition

$$M(\delta_i(s) - 1) \leq s_i - b_i < M\delta_i(s)$$

However, as computers don't really like strick inequalities we can change our condition with :

$$M(\delta_i(s) - 1) \leq s_i - b_i \leq M\delta_i(s) - \varepsilon$$

where ε is a small number.

Now that we have defined the boolean variables $\delta_i(s)$, we can use them to define continuous variables $w_i(s)$ for each course i and for each student s such that :

$$w_i(s) = \begin{cases} w_i, \text{ if } s_i \geq b_i \\ 0, \text{ otherwise} \end{cases}$$

We remind ourselves that the w_i are the weights attributed to course i and that it is our goal to find them.

In order to introduce the behavior of this new variable in our linear programming algorithm we can add the following conditions :

$$\begin{cases} w_i \geq w_i(s) \geq 0 \\ \delta_i(s) \geq w_i(s) \geq \delta_i(s) + w_i - 1 \end{cases}$$

Now, the only thing left to define for our linear programming algorithm is the function to maximize. Here we consider the "prediction margin" for each student s noted σ_s such that :

$$\begin{cases} \forall s \in A, & \sum w_i(s) - \lambda - \sigma_s = 0 \\ \forall s \in R, & \sum w_i(s) - \lambda + \sigma_s = -\varepsilon' \end{cases}$$

where ε' is another small positive number.

σ_s is such that it is the positive difference between the student's sum of weight and the margin. That means that in order to have a good model and find good w and λ, we must maximize the sum of all σ_s.

To sum up, the linear programming algorithm that will allow us to find the weight vector w and the threshold λ is described by :

$$\mathrm{Max} \sum_s \sigma_s$$

$$\begin{array}{ll} \text{subject to} & M(\delta_i(s) - 1) \leq s_i - b_i \leq M\delta_i(s) - \varepsilon \\ & w_i \geq w_i(s) \geq 0 \\ & \delta_i(s) \geq w_i(s) \geq \delta_i(s) + w_i - 1 \\ & \forall s \in A, \quad \sum w_i(s) - \lambda - \sigma_s = 0 \\ & \forall s \in R, \quad \sum w_i(s) - \lambda + \sigma_s = -\varepsilon' \\ & \\ \text{where} & \delta_i(s) \in \{0,1\} \\ & w_i(s) \in [0,1] \\ & \sigma_s \in [0,1] \\ & w_i \in [0,1] \\ & \lambda \in [0,1] \end{array}$$

3.2 SAT Solver Method

Now, let's take a different approach. Rather than using the linear programming algorithm method, let's use the SAT-solver method [8] [7].

Two classes

Let α_{ki} a boolean variable that is true if and only if the evaluation k on criterion i is above the frontier and β_C a boolean variable that is true if and only if the coalition of criteria is a majority.

To ensure that every evaluations that are ranked above a "good" evaluation are also good, our SAT-solver should satisfy the following clause :

$$\forall k' > k, \quad \alpha_{ki} \Rightarrow \alpha_{k'i} \text{ i.e. } \neg\alpha_{ki} \vee \alpha_{k'i}$$

In the same way, if one coalition of criteria is a majority, all coalitions of criteria that include it should also be majorities.

$$\forall C \subset C', \quad \beta_C \Rightarrow \beta_{C'} \text{ i.e. } \neg\beta_C \vee \beta_{C'}$$

Now, for all students that are accepted, we should write clauses that state it. Those clauses are in the form :

$$\forall C \subset \mathcal{N}, \quad \bigwedge_{i \in C} \neg\alpha_{ki} \Rightarrow \beta_{\mathcal{N} \setminus C} \text{ i.e. } \bigvee_{i \in C} \alpha_{ki} \vee \beta_{\mathcal{N} \setminus C}$$

This means that the criteria in the coalition C are not necessary to be a majority.

In the same way, for all students that are rejected, we also should write clauses stating it. Those clauses are in the form :

$$\forall C \subset \mathcal{N}, \quad \bigwedge_{i \in C} \alpha_{ki} \Rightarrow \neg\beta_C \text{ i.e. } \bigvee_{i \in C} \neg\alpha_{ki} \vee \neg\beta_C$$

This means that the in the C are not sufficient to be a majority.

Arbitrary number of classes

Now, let's get into the case where there are no longer two end classes (A and R) but an arbitrary number (H). We can derive a SAT formulation of this problem from the simpler one we described before [3] [4]. Let the index h describe the end profile index and α_{kih} the boolean variable that is true if on criterion i, the value k is sufficient at level h.

We can adapt the four clauses :

The ascending scales clause :

$$\forall k' > k, \quad \alpha_{kih} \Rightarrow \alpha_{k'ih} \text{ i.e. } \neg\alpha_{kih} \vee \alpha_{k'ih}$$

The strength clause :

$$\forall C \subset C', \quad \beta_C \Rightarrow \beta_{C'} \text{ i.e. } \neg\beta_C \vee \beta_{C'}$$

The outranking of alternatives by boundary below them :

$$\forall C \subset \mathcal{N}, \quad \forall h \in H, \quad \bigvee_{i \in C} \alpha_{a_i ih} \vee \beta_{\mathcal{N}\setminus C}$$

The outranking of alternatives by boundary above them :

$$\forall C \subset \mathcal{N}, \quad \forall h \in H, \quad \bigvee_{i \in C} \neg\alpha_{u_i ih} \vee \neg\beta_C$$

And finally we add another clause that will define the hierarchy of profiles amongst the different end results h.

$$\forall h' > h, \quad \neg\alpha_{kih} \Rightarrow \neg\alpha_{kih'} \text{ i.e. } \alpha_{kih} \vee \neg\alpha_{kih'}$$

MaxSAT solver

The two previous methods can give a model that can predict the class of each line of the data set (so for each student). However, when adding noise (some student's are assigned to random classes), we can see that there is no way for our normal conjunctive form to be satisfiable. Therefore, we must find a way around to take into account the possible noise in the data, hence the MaxSAT solver solution.

The MaxSAT solver solution transforms the decision problem of the SAT problem to an optimization problem [6] where we aim at maximizing the number of clause we satisfy. Specifically, as we are going to put weights on each one of our clause, the MaxSAT algorithm goal will be to maximize the sum of the weights of the satisfied clauses.

In this specific case, we want the model produced by the MaxSAT solver to follow a NCS model. Therefore, we want to give more importance to the "structural" clauses (the ascending scale clause, the coalitions strength clause and the hierarchy of profile clause) and give some slack around the "student" clauses (the outranking of alternatives by boundary below/above them clause). To achieve this hierarchy of clauses, we will put a weight W to each structural clauses and a weight w to each student clauses such that $W >> w$, more precisely, $W > n_{std\_clauses} w$ where n_{std_clause} is the number of student clauses. This will ensure that our algorithm puts a priority in satisfying the structural clause first and then the student ones [1].

4 Performances

4.1 Performances on test learning sets

Method	Learning set	accuracy	f1-score	time (s)
MILP	data6crit50e	100%	100%	0.81
	data6crit75ex	100%	100%	1.15
	data6crit100ex	100%	100%	1.19
SAT	data6crit50e	100%	100%	0.06
	data6crit75ex	100%	100%	0.08
	data6crit100ex	100%	100%	0.05
MaxSAT	data6crit50e	100%	100%	0.09
	data6crit75ex	100%	100%	0.06
	data6crit100ex	100%	100%	0.05

Our three implementations work very well on the three given example datasets. The scores are perfect, the models are able to perfectly find a solution, and the running times are also correct, even though the MILP is quite slower than both SAT formulations. Especially since with 6 criteria, 2 classes, no noise and 50 to 100 examples, the data remains easily exploitable with this volume.

4.2 The learning capacity is exact for SAT and MaxSAT and approximate for MILP

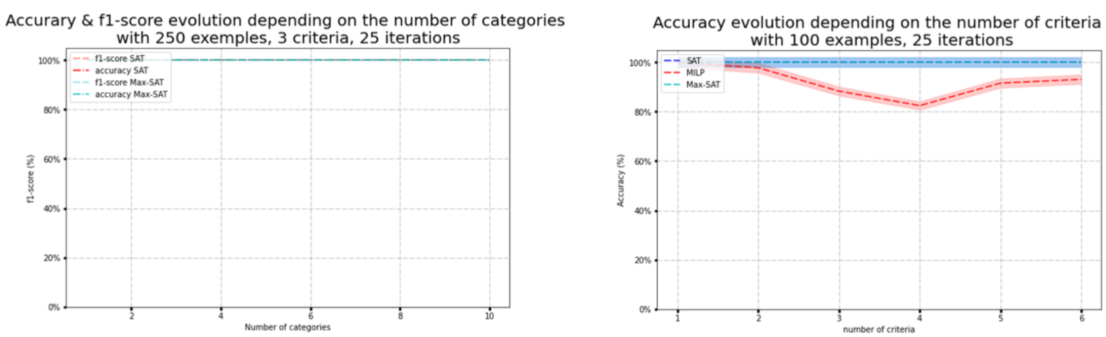

Figure 1: Accuracy and f1 score for one class and for multiclasses

Results interpretations : SAT and MaxSAT formulations are exact. MILP is approximate. SAT and MaxSAT formulations are also exact for multiclass.

4.3 The time evolution according to the numbers of criteria shows MILP exploding and SAT remaining the fastest model.

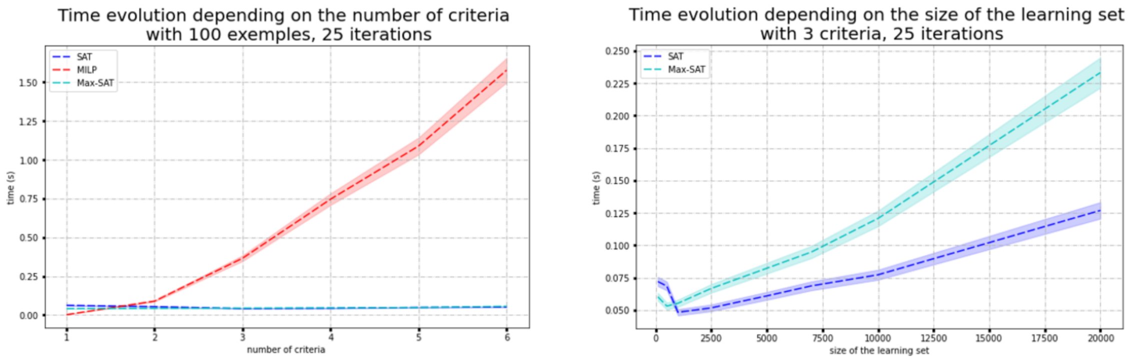

Figure 2: Time evolution according to the numbers of criteria and to the size of the learning set.

Results interpretations : MILP explodes in runtime with increased parameters. SAT formulation remains the fastest.

4.4 Noisy data impact all models, MaxSAT has the best results

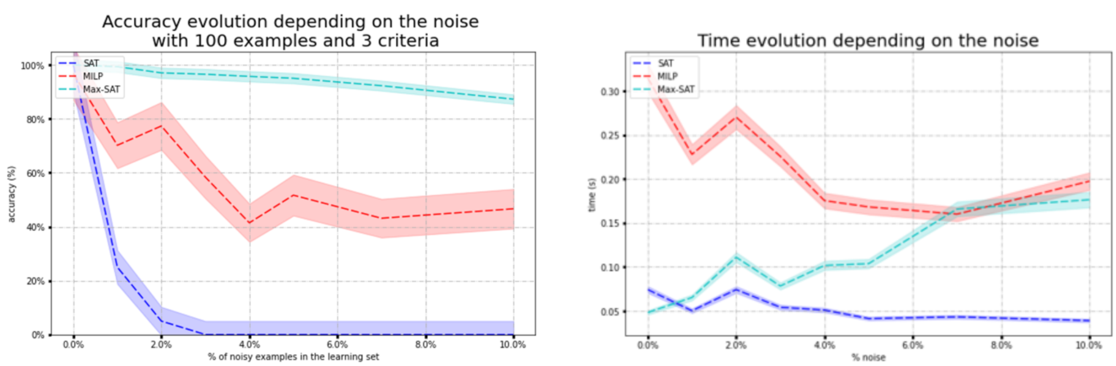

Figure 3: Accuracy, f1-score and time evolution according to the percentage of noise

Results interpretations : We observe MaxSAT formulation best fits the noisy data in order to find an NCS model that is as close as possible to real data. Also, MaxSAT runtime increases a little but remains lower than MILP runtime.

5 Conclusion

MILP	SAT	MaxSAT
Easily interpretable Takes into account noisy data Maintains good performance	More complex in formulation The fastest Perfect score (if perfect data set)	Perfect score (if perfect data set) Takes into account noisy data Keeps very good performances
Slow Approached Not implemented in multi-class	Does not converge if there is noise	A little slower than the SAT

In view of the global results the best solver seems to be the formulation in case of suspected imperfect data, otherwise the SAT formulation gives the best results the fastest.

Glossary

Inv-NCS Inverse Non-Compensatory Sorting problem takes as input a set of assignment examples, and computes (whenever it exists) an NCS sorting model which is consistent with this preference information.. 1

MaxSAT Maxium Satisfiability . 1, 4–6

MILP Mixed-integer linear programming . 1, 5, 6

MR-Sort Majority Rule Sorting. 1

NCS Non compensatory sorting, The DM expresses preferences from which a specific NCS model is inferred. 1, 4, 6

SAT Solve the Boolean satisfiability problem. 1, 4–7

References

[1] O. Khaled V. Mousseau W. Ouerdane A. Tlili, K. Belahcène. , learning non-compensatory sorting models using efficient sat/maxsat formulations. *European Journal of Operational Research*, August 2021.

[2] Vincent Mousseau Agnes Leroy and Marc Pirlot. Learning the parameters of a multiple criteria sorting method. *Algorithmic Decision Theory*, 229:219–233, 2011.

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

[4] T. Marchant D. Bouyssou. An axiomatic approach to noncompensatory sorting methods in mcdm, ii: More than two categories. *European Journal of Operational Research*, 178(1):246–276, 2007.

[5] Marc Pirlot Eda Ersek Uyanik, Vincent Mousseau and Olivier Sobrie. Enumerating and categorizing positive boolean functions separable by a k-additive capacity. *Discrete Applied Mathematics*, 229:17–30, 2017.

[6] R. Martins J. Berg, M. Järvisalo. Advances in maximum satisfiability. *ECAI'20 Online*, September 4, 2020.

[7] N. Maudet V. Mousseau W. Ouerdane K. Belahcene, C. Labreuche and Y. Chevaleyre. Accountable approval sorting. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[8] N. Maudet V. Mousseau W.Ouerdane K. Belahcene, C. Labreuche. An efficient sat formulation for learning multiple criteria non-compensatory sorting rules from examples. *Computers Operations Research*, 97:58–71, 2018.