

# System de décision

Trois approches pour  
résoudre le problème  
d'INV-NCS

Ariane Dalens

Magali Morin

Lucas Sor



CentraleSupélec

# Sommaire

---

## 01. Introduction

## 02. Explications théoriques

- a. Algorithme de programmation linéaire
- b. SAT Solver
- c. MaxSAT Solver

## 03. Performances

## 04. Conclusion

# Introduction

---

## Situation

Un nouveau jury d'un programme d'éducation de haut niveau doit décider de l'admission de ses étudiants selon leurs notes dans 4 cours différents : mathématiques, physiques, littérature et histoire. Ils disposent des notes et des status d'admission des élèves de l'année dernière, comment doivent-il procéder ?

Résoudre le problème Inv-NCS a été prouvé NP-difficile<sup>[6]</sup>. Pour résoudre ce problème, nous avons implémenté trois approches : Mixed-Integer Linear programming<sup>[7]</sup>, SAT<sup>[5]</sup> and Max-SAT<sup>[9]</sup>.



# Sommaire

---

**01.** Introduction

## **02.** Explications théoriques

- a. Algorithme de programmation linéaire
- b. SAT Solver
- c. MaxSAT Solver

**03.** Performances

**04.** Conclusion

# Explications théoriques

## Mixed-Integer Linear Programming (MILP) with MaxSum optimum

Le vecteur de poids  $w$  et le seuil  $\lambda$  sont décrits par :

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

where

$$\begin{array}{l} \delta_i(s) \in \{0, 1\} \\ w_i(s) \in [0, 1] \\ \sigma_s \in [0, 1] \\ w_i \in [0, 1] \\ \lambda \in [0, 1] \end{array}$$

L'étudiant  $s$  valide-t-il le cours  $i$  ?

Si oui, on ajoute le poids du cours

Et on définit la marge de l'étudiant

# Explications théoriques

---

**SAT Solver : Deux classes/mentions (accepté et refusé).**

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

Croissance des notes

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

Majorité des coalitions

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

Étudiants acceptés

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

Étudiants refusés

# Explications théoriques

---

**SAT Solver : Nombre arbitraire de mentions/classes H.**

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

Croissance des notes

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

Majorité des coalitions

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

Comparaison des alternatives avec la frontière inférieure

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

Comparaison des alternatives avec la frontière supérieure

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

Hiérarchie des mentions/classes

# Explications théoriques

---

**MaxSAT Solver:** Nous formulons le problème d'optimisation relaxé consistant à trouver le sous-ensemble d'exemples d'apprentissage correctement restaurés de cardinalité maximale avec une approche de contrainte souple, en utilisant la formulation de weighted-MaxSAT [9].

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

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

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

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

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



# Sommaire

---

**01.** Introduction

**02.** Explications théoriques

- a. Algorithme de programmation linéaire
- b. SAT Solver
- c. MaxSAT Solver

**03.** Performances

**04.** Conclusion

# Benchmark

## Learning set de test

---

MacOS X 12.1 - Intel Core i5 CPU @3.1 GHz  
8Go de RAM  
Gophersat v1.3  
Python3.8  
[decision-system-project](#)



Nos trois implémentations marchent parfaitement sur les trois jeu de données d'exemple donnés.

- 6 critères
- 2 classes
- 50 à 100 exemples

Les données restent facilement exploitables avec ce volume là.

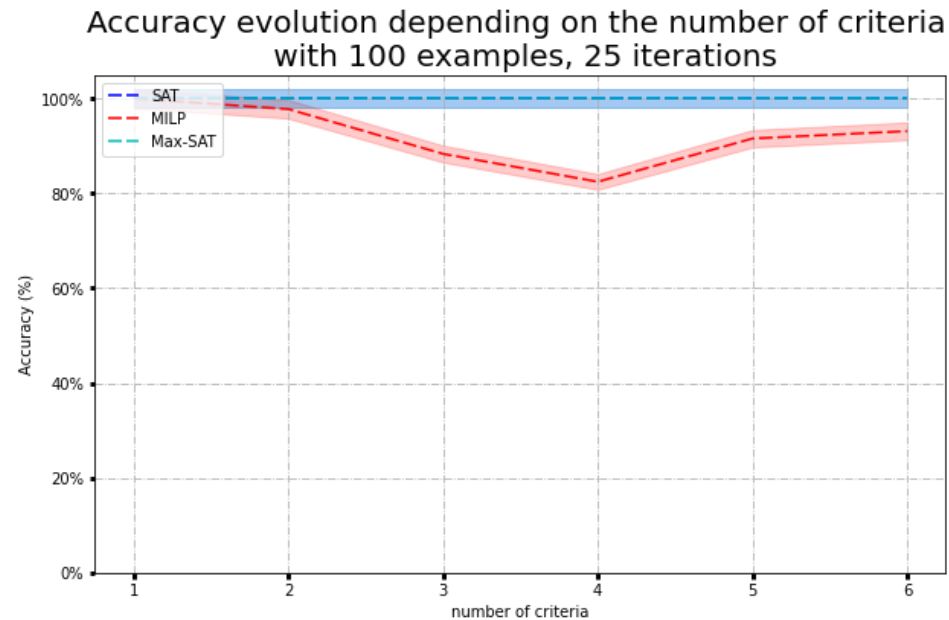
Method	Learning set	accuracy	f1-score	time (s)
MILP	data6crit50e	100%	100%	0.81
	data6crit75ex	100%	100%	1.15
	data6crit100ex	100%	100%	1.19
SAT	data6crit50e	100%	100%	0.06
	data6crit75ex	100%	100%	0.08
	data6crit100ex	100%	100%	0.05
MaxSAT	data6crit50e	100%	100%	0.09
	data6crit75ex	100%	100%	0.06
	data6crit100ex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

# Performances

## Capacité d'apprentissage

MacOS X 12.1 - Intel Core i5 CPU @3.1 GHz  
8Go de RAM  
Gophersat v1.3  
Python3.8  
[decision-system-project](#)

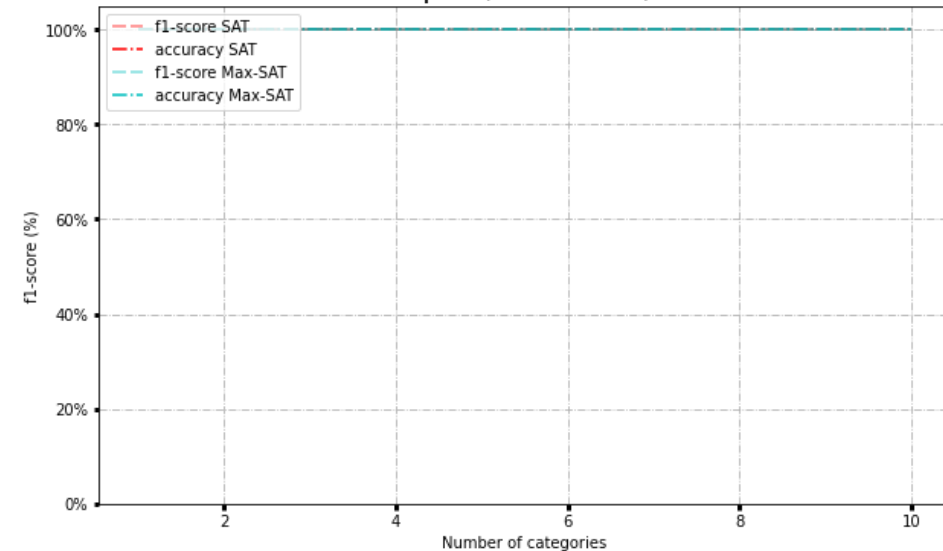
### Résultats pour une classe:



Formulations SAT/MaxSAT exactes  
MILP approché

### Résultats en multi-classe:

Accuracy & f1-score evolution depending on the number of categories with 250 examples, 3 criteria, 25 iterations



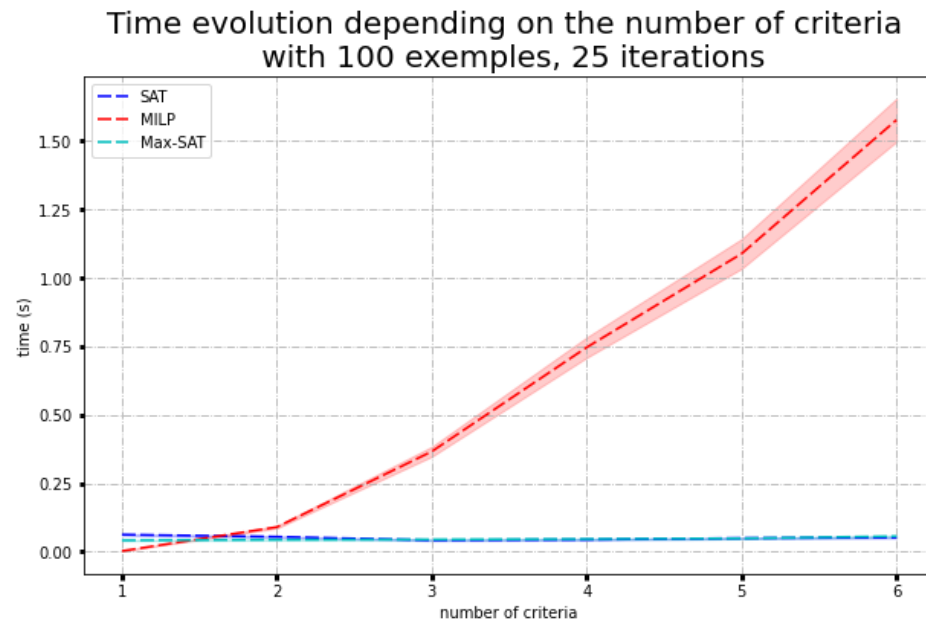
Formulations SAT/MaxSAT exactes en  
multiclass

# Performances

## Temps de calcul

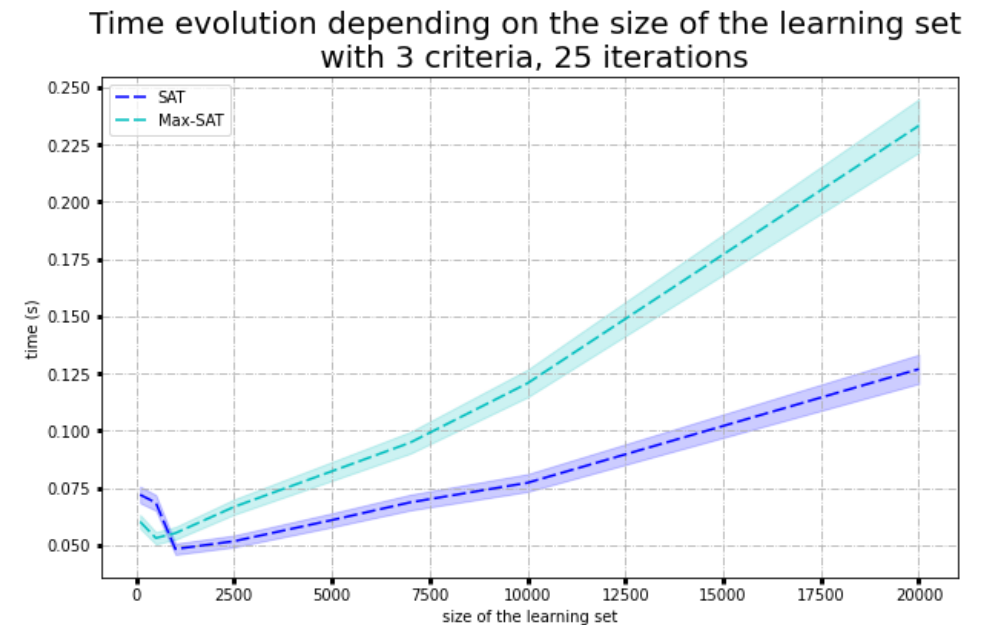
MacOS X 12.1 - Intel Core i5 CPU @3.1 GHz  
8Go de RAM  
Gophersat v1.3  
Python3.8  
[decision-system-project](#)

### Résultats en fonction du nombre de critères:



MILP explose en runtime avec l'augmentation des paramètres

### Résultats en fonction de la taille du learning set:

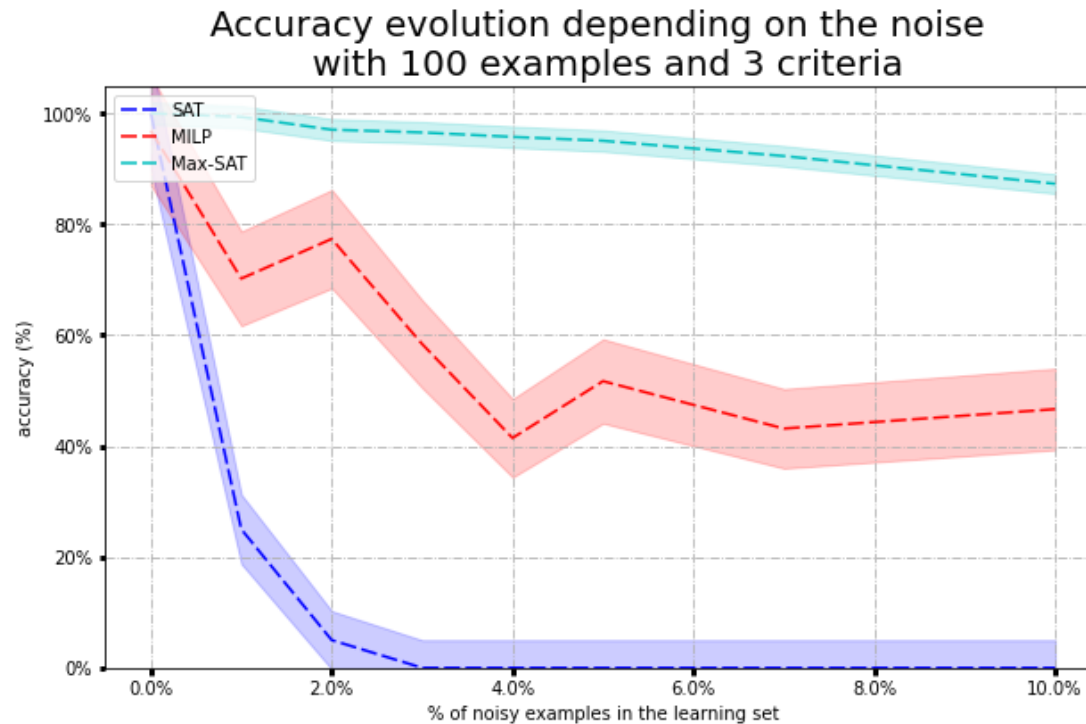


Formulation SAT reste la plus rapide

# Performances

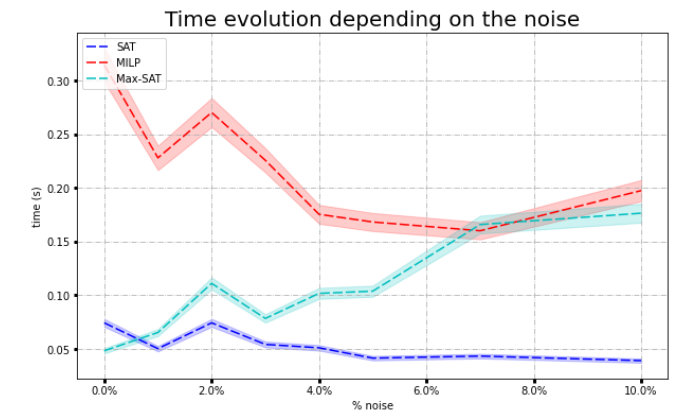
## Adaptation à des données bruitées

MacOS X 12.1 - Intel Core i5 CPU @3.1 GHz  
8Go de RAM  
Gophersat v1.3  
Python3.8  
decision-system-project



La formulation **MaxSAT** est celle qui permet de s'adapter au mieux au données bruitées afin de trouver un modèle NCS qui soit le plus proche de la réalité possible.

Avec des données bruitées, la durée du **MaxSAT** augmente un peu mais reste inférieure à celle du MILP



# Sommaire

---

## 01. Introduction

## 02. Explications théoriques



- a. Algorithme de programmation linéaire
- b. SAT Solver
- c. MaxSAT Solver

## 03. Performances

## 04. Conclusion

# Conclusion

---

	MILP	SAT	MaxSAT
	<ul style="list-style-type: none"><li>Facilement <b>interprétable</b></li><li>Prend en compte <b>les données bruitées</b></li><li>Garde de bonnes performances</li></ul>	<ul style="list-style-type: none"><li>Plus complexe dans la formulation</li><li>Le plus rapide</li><li><b>Score parfait</b> (si jeu de données parfait)</li></ul>	<ul style="list-style-type: none"><li><b>Score parfait</b> (si jeu de données parfait)</li><li>Prend en compte <b>les données bruités</b></li><li>Garde de très bonnes performances</li></ul>
	<ul style="list-style-type: none"><li><b>Lent</b></li><li>Approché</li><li><b>Non implémenté en multi-classe</b></li></ul>	<ul style="list-style-type: none"><li>Ne <b>converge pas</b> s'il y a du <b>bruit</b></li></ul>	<ul style="list-style-type: none"><li>Un peu plus <b>lent</b> que le SAT</li></ul>

# Références

---

- [1] D. Bouyssou, T. Marchant, 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).
- [2] D. Bouyssou, T. Marchant, An axiomatic approach to noncompensatory sorting methods in MCDM, II: More than two categories, European Journal of Operational Research, 178(1):246–276, (2007).
- [3] Eda Ersek Uyanik, Vincent Mousseau, Marc Pirlot, and Olivier Sobrie. Enumerating and categorizing positive boolean functions separable by a k-additive capacity. Discrete Applied Mathematics, 229:17-30, (2017).
- [4] Agnes Leroy, Vincent Mousseau, and Marc Pirlot. Learning the parameters of a multiple criteria sorting method. Algorithmic Decision Theory, 219-233, (2011).
- [5] Belahcene K., Labreuche C., Maudet N., Mousseau V., and Ouerdane, W. An efficient SAT formulation for learning multiple criteria non-compensatory sorting rules from examples, Computers & Operations Research, 97, 58–71, (2018).
- [6] K. Belahcene, C. Labreuche, N. Maudet, V. Mousseau, W. Ouerdane, and Y. Chevaleyre, ‘Accountable approval sorting’, in Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), (2018).
- [7] A. Leroy, V. Mousseau, and M. Pirlot, ‘Learning the parameters of a multiple criteria sorting method’, in International Conference on Algorithmic Decision Theory, pp. 219–233. Springer, (2011).
- [8] J. Berg, M. Järvisalo, R. Martins, Advances in Maximum Satisfiability, ECAL’20 Online, September 4, 2020.
- [9] A. Tlili, K. Belahcène, O. Khaled, V. Mousseau, W. Ouerdane, Learning Non-Compensatory Sorting models using efficient SAT/MaxSAT formulations, European Journal of Operational Research, August 2021



# Appendix

---

# Glossaire

---

**NCS:** Non compensatory sorting, The DM expresses preferences from which a specific NCS model is inferred

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

**MR-Sort:** Majority Rule Sorting

**MILP:** Mixed-integer linear programming

**SAT:** solve the Boolean satisfiability problem

**MaxSAT:** Maximum Satisfiability