



FACULTE DES SCIENCES ET INGENIERIE

MASTER INFORMATIQUE

**Projet ANDROIDE**

**Interface web pour systèmes de recommandation en  
optimisation combinatoire multi-critères**

ETUDIANTS

Yuhan WANG

Tianyu WANG

ENCADRANT

Thibaut LUST

June 8, 2020

# Contents

Introduction	4
Ressources	4
<b>I Méthode d'optimisation &amp; Définitions mathématiques</b>	<b>5</b>
1 Problème d'optimisation multi-critères	5
2 Problème d'optimisation combinatoire	5
3 Méthode Minimax Regret	6
3.1 Pairwise Max Regret (PMR)	6
3.2 Max Regret (MR)	6
3.3 Minimax Regret (MMR)	6
4 Elicitation incrémentale basée sur le regret	6
5 Un exemple	6
6 Premier implémentation en Python	9
6.1 Les fonctions	9
6.2 Test	10
<b>II Interface web</b>	<b>11</b>
7 Organisation des tâches	11
7.1 Etape de développement	11
7.2 Calendrier	12
8 Développement technique	12
8.1 Signaux et Slot	13
9 Pages et fonctionnalités du produit	13
9.1 Le modèle mathématique pour les voitures	14
9.2 Landing page	14
9.3 Quiz time	14
9.4 Solution optimale personnalisée	16
10 Organisation du code	17
<b>III Annexes</b>	<b>20</b>

<b>11 Gurobi Optimizer</b>	<b>20</b>
11.1 Présentation . . . . .	20
11.2 Fonctions principales . . . . .	20
<b>12 Qt et Qt Designer</b>	<b>20</b>
12.1 Fonctions principales Qt . . . . .	20
12.2 QSS et Qt Designer . . . . .	21

## Introduction

De nos jours, le problème de la décision multi-critères auquel on fait face dans la vie réelle recouvre tous les domaines. Ce problème consiste à choisir, en présence de critères multiples, une alternatives optimale parmi un nombre d'alternatives. L'optimisation combinatoire multi-critères, autrement dit combinatoire "multi-objectif" est donc un domaine fondamental de l'aide à ce problème.

Non seulement dans l'économie et dans l'industrie, l'optimisation combinatoire multi-critères a aussi un intérêt grandissant au quotidien. De nombreux scientifiques doivent se faire face à déterminer la solution correspondant mieux aux préférences du décideur. Peut-être ça explique pourquoi l'application des systèmes de recommandation est si répandue. Les systèmes de recommandation sont une forme scientifique de filtrage visant à présenter le ou les élément(s) qui est(sont) susceptibles d'intéresser l'utilisateur. Ils en existent dans Youtube, Spotify, Facebook, Instagram en utilisant différentes algorithmes de recommandation. Parfois ils analysent la préférence d'un utilisateur par les caractéristiques des objets qu'il aime et lui proposer les objets similaires, parfois ils demandent aux utilisateurs de compléter une enquête dans l'étape d'inscription pour détailler son portrait.

C'est dans ce contexte que s'inscrit notre projet. Ce qu'on voudrait réaliser, c'est une interface en ligne pour les individus confrontés aux problèmes d'optimisation multi-critères. Cette interface sera capable d'interagir avec l'utilisateur, et de modéliser ses préférences en utilisant la méthode basée sur l'élicitation incrémentale de préférence et le MiniMax Regret.

## Ressources

Codes: <https://github.com/yuhanWG/Projet-M1.git/tree/master/version-final-pandroide2>

## Part I

# Méthode d'optimisation & Définitions mathématiques

## 1 Problème d'optimisation multi-critères

Dans sa forme la plus générale, un problème d'optimisation multi-critère(ou multiobjectif) consiste à trouver dans un ensemble des admissibles, une solution ou un sous-ensemble de solutions optimisant des objectifs.

Considérons un problème d'optimisation multi-critères, nous notons  $\mathcal{X}$  un ensemble d'alternatives (produits, candidats, objets,...) où chaque alternative est caractérisée par un vecteur composé de  $N = \{1, 2, 3, \dots, n\}$  critères à évaluer.

$$x \in \mathcal{X}, x = \{x_1, x_2, \dots, x_n\}$$

Pour comparer les utilités des alternatives, il nous faut une fonction d'agrégation  $f_w$  qui transforme le vecteur de performance en une valeur. Il existe plusieurs formes pour la fonction d'agrégation, nous avons utilisé la somme pondérée où  $w = \{w_1, w_2, \dots, w_n\}$  représente le vecteur des poids de préférence pour les critères :

$$f_w(x) = \sum_{i=1}^n w_i x_i, x \in \mathcal{X}$$

$$\sum_{i=1}^N w_i = 1$$

Pour simplifier la construction du problème, on considère que c'est un problème de minimisation et on transforme les valeurs à maximiser en valeurs négatives. Donc:

$$x \text{ est préféré à } x' \equiv f_w(x) \leq f_w(x')$$

## 2 Problème d'optimisation combinatoire

Dans sa forme la plus générale, un problème combinatoire(ou l'optimisation discrète) consiste à trouver la meilleure solution parmi un ensemble des solutions réalisables. C'est un problème facile en théorie: on peut calculer et comparer toutes les solutions et en prendre la meilleure. Mais cet ensemble ne saurait être décrit par une liste exhaustive car généralement en pratique, le nombre des solutions réalisables rend son énumération impossible.

Un exemple fameux pour le problème d'optimisation combinatoire est **le problème du voyageur de commerce** où étant donné une liste des villes et les distances entre eux, un commerçant a besoin de parcourir les toutes les villes en effectuant le trajet le plus court possible. Au lieu calculer tous les trajets possibles(ça nécessite  $\frac{(N-1)!}{2}$  fois calculs), nous préférons plutôt à chercher partie par partie.

### 3 Méthode Minimax Regret

Le **Regret** peut être mesuré comme la différence d'utilité entre la décision proposée et la décision optimale. Par exemple, si la solution optimale pour l'utilisateur est  $x$  mais notre système de recommandation lui propose  $y$  au lieu, le regret dans ce cas est égale à  $f_w(x) - f_w(y)$ . Quand on prend la meilleure décision, le regret doit être 0. L'idée principale de la méthode Minimax Regret est de minimiser le regret dans le pire cas.

#### 3.1 Pairwise Max Regret (PMR)

PMR représente le pire regret parmi tous les couples possibles  $(x, x') \in \mathcal{X}^2$ .  $\theta$  représente les informations disponibles sur la préférence de l'utilisateur entre deux alternatives et  $\Omega_\theta$  tous les jeux de poids  $w$  compatibles avec  $\theta$ .

$$\Omega_\theta = \{w : \forall (a, b) \in \theta, f_w(a) \leq f_w(b)\}$$

$$PMR(x, x', \Omega_\theta) = \max_{w \in \Omega_\theta} \{f_w(x) - f_w(x')\}$$

#### 3.2 Max Regret (MR)

$MR(x, \mathcal{X}, \Omega_\theta)$  est le pire cas de choisir l'alternative  $x$  au lieu d'une autre solution  $x' \in \mathcal{X}$ .

$$MR(x, \mathcal{X}, \Omega_\theta) = \max_{x' \in \mathcal{X}} PMR(x, x', \Omega_\theta)$$

#### 3.3 Minimax Regret (MMR)

Une solution  $x^* \in \mathcal{X}$  est optimale si elle atteint une valeur de regret minimale, c'est-à-dire:

$$x^* \in \operatorname{argmin}_{x \in \mathcal{X}} MR(x, \mathcal{X}, \Omega_\theta)$$

$$MMR(\mathcal{X}, \Omega_\theta) = \min_{x \in \mathcal{X}} MR(x, \mathcal{X}, \Omega_\theta)$$

Dans ce cas, recommander une telle solution garantie que le pire regret est minimal.

## 4 Elicitation incrémentale basée sur le regret

L'idée de l'élicitation incrémentale utilisée dans notre projet est de demander progressivement une suite des requêtes qui contient la MMR-alternative et son pire adversaire au décideur afin de réduire l'incertitude sur le vecteur des poids modélisant les préférences du décideur.

## 5 Un exemple

La fonction d'agrégation retourne une utilité pour chaque alternative, ici pour simuler un décideur virtuel, nous supposons le jeux de poids caché  $w^c = (0.326851, 0.318176, 0.354973)$ . Considérons l'exemple suivant comprenant 5 alternatives et 3 critères :

	1	2	3
$x_1$	32	42	26
$x_2$	26	45	35
$x_3$	39	43	18
$x_4$	45	23	36
$x_5$	37	27	37

### Itération numéro 1:

1. On calcule la matrice des PMRs avec l'aide de Gurobi Solver. Maintenant on n'a aucune contrainte dans le programme linéaire, donc le  $w$  qui maximise  $PMR(x, x') = \max_{w \in \Omega_\theta} \{f_w(x_c) - g_w(x'_c)\}$  est choisi parmi  $(1,0,0), (0,1,0)$  et  $(0,0,1)$ . En posant une suite des questions, ce  $w$  va s'approcher et à la fin être proche de  $w^c$  (le jeu de poids caché utilisé pour simuler les réponses du décideur).

	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
$x_1$	0.00	6.00	8.00	19.00	15.00
$x_2$	9.00	0.00	17.00	22.00	18.00
$x_3$	7.00	13.00	0.00	20.00	16.00
$x_4$	13.00	19.00	18.00	0.00	8.00
$x_5$	11.00	11.00	19.00	4.00	0.00

2. On détermine les regrets maximums pour chaque alternative.

	MR
$x_1$	19
$x_2$	22
$x_3$	20
$x_4$	19
$x_5$	19

3.  $\text{Minimax}\{MR\} = \{x_1, x_4, x_5\}$ , on en prend  $x_1$ , on peut aussi en sélectionner une hasard.
4. On sélectionne le pire adversaire, celui qui donne le regret maximal quand on choisit  $x_1$  au lieu de lui, c'est  $x_4$ .
5. On demande au décideur de comparer  $x_1$  et  $x_4$  en lui posant la question: "est-ce que vous préférez  $x_1$  ou  $x_4$ ?". Ici on utilise  $w^c$  pour répondre sans avoir un vrai décideur :

$$f_w(x_1) = 33.05 < f_w(x_4) = 34.81$$

6. On ajoute une contrainte dans le programme linéaire :

$$32w_1 + 42w_2 + 26w_3 \leq 45w_1 + 23w_2 + 36w_3$$

### Itération numéro 2:

1. Comme  $x_1$  est préféré à  $x_4$ , on l'enlève, la nouvelle matrice des PMR est calculées à l'aide du solveur de programmation linéaire Gurobi :

	$x_1$	$x_2$	$x_3$	$x_5$
$x_1$	0.000000	6.000000	8.000000	3.125000
$x_2$	9.000000	0.000000	17.000000	4.896552
$x_3$	7.000000	13.000000	0.000000	7.687500
$x_5$	11.000000	11.000000	19.000000	0.000000

2. On détermine les regrets maximums pour chaque alternative :

	MR
$x_1$	8
$x_2$	17
$x_3$	13
$x_5$	19

3. On sélectionne une solution de minimax regret, c'est  $x_1$ . Son pire adversaire, c'est  $x_3$ .
4. On demande au décideur de comparer  $x_1$  et  $x_3$ . Le décideur répond qu'il préfère  $x_3$  (on simule sa réponse grâce au poids caché). En effet:

$$f_w(x_1) = 33.05 > f_w(x_3) = 32.82$$

5. On ajoute la contrainte et on enlève  $x_1$ :

$$39w_1 + 43w_2 + 18w_3 \leq 32w_1 + 42w_2 + 26w_3$$

### Itération numéro 3:

1. On calcule la matrice des PMRs :

	$x_2$	$x_3$	$x_5$
$x_2$	0.000000	17.000000	4.896517
$x_3$	-1.000000	0.000000	0.636364
$x_5$	6.800000	19	0.000000

2. On détermine les regrets maximums pour chaque alternative :

	MR
$x_2$	17
$x_3$	0.64
$x_5$	19

3. On demande au décideur de comparer  $x_3$  et  $x_5$ . Le décideur répond qu'il préfère  $x_3$  (on simule sa réponse grâce au poids caché). En effet:

$$f_w(x_5) = 33.82 > f_w(x_3) = 32.82$$

4. On ajoute la contrainte :

$$39w_1 + 43w_2 + 18w_3 \leq 37w_1 + 27w_2 + 37w_3$$

### Itération finale :

On calcule les regrets avec une solution  $x_3$ , elle a un regret nul et son pire adversaire est elle-même. La matrice des PMRs est:

	$x_2$	$x_3$
$x_2$	0.000000	17.000000
$x_3$	-1.000000	0.000000

Donc c'est la solution optimale. La procédure s'arrête et on recommande la solution  $x_3$  au décideur.

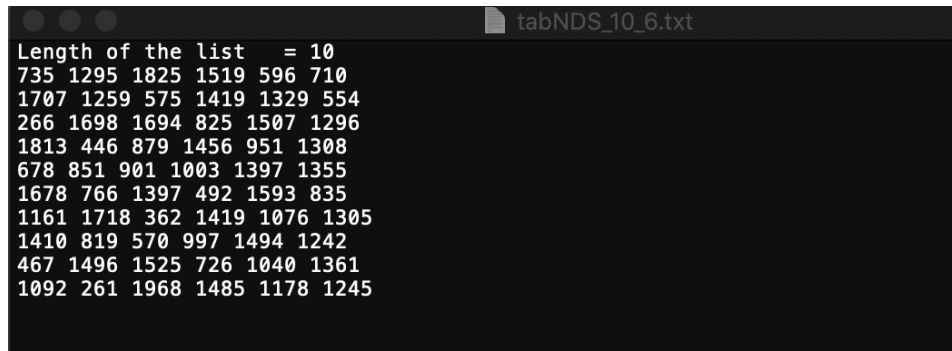


## 6 Premier implémentation en Python

Avant la création de l'interface, la première étape importante de notre projet est de simuler la méthode **MiniMax Regret** en Python. Dans cette partie, nous détaillons d'abord les fonctions qu'on a écrites pour réaliser la méthode et après on analyse les résultats obtenus en testant l'algorithme avec une vingtaine d'exemples.

### 6.1 Les fonctions

Les exemples de données sont sous la forme de .txt, chaque ligne représente un alternative et chaque colonne enregistre les performances des alternatives pour un critère. La figure ci-dessous montre une base de données composée de 10 alternatives, 6 critères.



```
Length of the list = 10
735 1295 1825 1519 596 710
1707 1259 575 1419 1329 554
266 1698 1694 825 1507 1296
1813 446 879 1456 951 1308
678 851 901 1003 1397 1355
1678 766 1397 492 1593 835
1161 1718 362 1419 1076 1305
1410 819 570 997 1494 1242
467 1496 1525 726 1040 1361
1092 261 1968 1485 1178 1245
```

Figure 1: Un exemple de données pour le test

Les fonctions principaux qu'on a utilisées pour simuler la méthode MiniMax Regret sont suivantes:

*readFile(nomfile)*: lire les fichiers de données et les transformer vers une matrice en Python pour faciliter les traitements suivants.

*poid\_aleatoire(nb)*: initialiser le poids cachées pour les critères.

*pmr(matrice)*: en prenant la matrice alternatives-critères, retourner la matrice des PMRs.

*new\_pmr(m,matrice)*: en prenant le modèle de programme linéaire, recalculer la matrice des PMRs.

*max\_regret(PMRs)*: retourner l'alternative MiniMax Regret et son pire adversaire.

*Query(poid,sol1,sol2)*: simuler la décision de l'utilisateur, en prenant le poids caché et deux solutions, comparer les deux. Retourner True si sol1 est préférée que sol2, False sinon.

---

**Algorithm 1** MiniMax Regret

---

```
Création d'un programme linéaire m;
matrice=readFile(file)
poids=poid(nb_c)
pmrs=pmr(matrice)
cpt=0

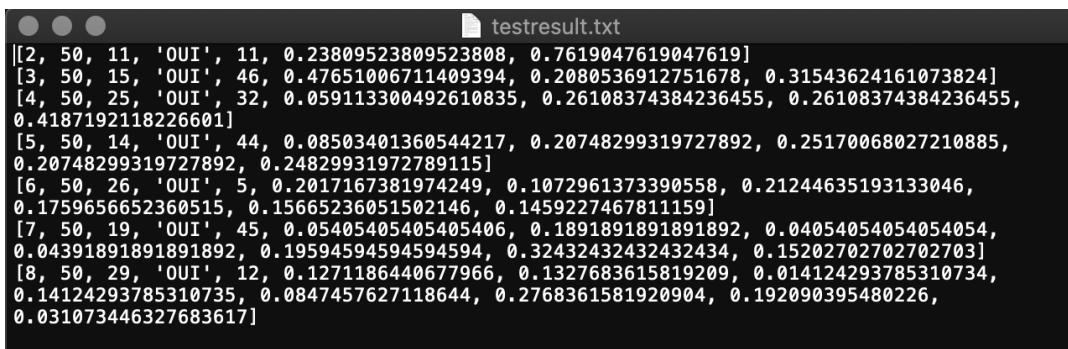
while (minimax Regret>0) do
  sol1,sol2 ← max_regret(pmrs)
  if Query(poids,sol1,sol2) then
    ajoute contrainte: sol1 meilleure que sol2;
    matrice ← matrice supprime sol2
  else
    ajoute contrainte: sol2 meilleure que sol1;
    matrice ← matrice supprime sol1
  end
  pmrs ← new_pmr(m,matrice)
  cpt ← cpt + 1
end
```

---

## 6.2 Test

Pour tester nos méthode, il nous faut un utilisateur qui fait son choix parmi les deux solutions. Pour simuler cette action du choix et vérifier la réponse, pour chaque exemple de test, on utilise la fonction *poid\_aleatoire(nb)* qui prend au hasard un ensemble de poids cachés dont la somme est égale à 1.

On teste la méthode MiniMax Regret et pour chaque exemple, on note dans le fichier testresult.txt leurs nombres critères, le nombre des alternatives, le nombre des questions posées, si la méthode propose la bonne solution, le numéro de solution optimale et les poids des critères. Cette figure ci-dessous représente les informations enregistrées après avoir testé les bases de données de 50 alternatives et le nombre de critère varie entre 2 et 9.



```
testresult.txt
[[2, 50, 11, 'OUI', 11, 0.23809523809523808, 0.7619047619047619]
[3, 50, 15, 'OUI', 46, 0.47651006711409394, 0.2080536912751678, 0.31543624161073824]
[4, 50, 25, 'OUI', 32, 0.059113300492610835, 0.26108374384236455, 0.26108374384236455,
0.4187192118226601]
[5, 50, 14, 'OUI', 44, 0.08503401360544217, 0.20748299319727892, 0.25170068027210885,
0.20748299319727892, 0.24829931972789115]
[6, 50, 26, 'OUI', 5, 0.2017167381974249, 0.1072961373390558, 0.21244635193133046,
0.1759656652360515, 0.15665236051502146, 0.1459227467811159]
[7, 50, 19, 'OUI', 45, 0.05405405405405406, 0.1891891891891892, 0.04054054054054054,
0.04391891891891892, 0.19594594594594594, 0.32432432432432434, 0.15202702702702703]
[8, 50, 29, 'OUI', 12, 0.1271186440677966, 0.1327683615819209, 0.014124293785310734,
0.14124293785310735, 0.0847457627118644, 0.2768361581920904, 0.192090395480226,
0.031073446327683617]
```

Figure 2: fichier testresult.txt

## Part II

# Interface web

Pour ce projet, le produit final prend la forme d'une interface qui permet aux individus confrontés à un problème d'optimisation combinatoire multi-critères d'avoir une aide en ligne. L'idée principale est de demander à l'utilisateur de comparer deux à deux des alternatives proposées par l'algorithme **MiniMax Regret** et en déduire une solution dit "optimale et personnalisée" selon le choix de l'utilisateur.

L'interface permet également aux utilisateurs de choisir les problèmes d'optimisation combinatoire multi-critère qui les intéressent. Comme dans le projet final, l'utilisateur peut choisir entre la base de données des voitures et celle des maisons.

Nous détaillons dans la suite quelles sont les grandes étapes de travail et de quelle manière il a été organisé.

## 7 Organisation des tâches

### 7.1 Etape de développement

Pendant le premier mois nous avons organisé des réunions hebdomadaires avec notre encadrant Thibaut Lust. Nous avons travaillé notamment sur les recherches documentaires pour mieux comprendre la méthode Minimax Regret. Nous avons aussi élaboré ensemble le cahier des charges et la recherche bibliographique. Après ce travail, nous nous sommes répartis les charges de la façon suivante :

#### Yuhan WANG

1. Etude théorique des algorithmes.
2. Implémentation du code en Python.
3. Amélioration de l'implémentation et de l'organisation du code pour être compatible avec les fonctionnalités d'interface.

#### Tianyu WANG

1. Etude des besoins de l'utilisateur.
2. Recherches sur les bases de données utilisables et intéressantes pour l'élicitation des préférences par la méthode MiniMax Regret.
3. Choix des langages et de l'environnement pour développer l'interface, analyse des avantages et des inconvénients.
4. Création, design et amélioration de l'interface.

## 7.2 Calendrier

- Lundi 27 janvier : début du semestre et début du projet
- Février : prise en main du projet, implémentation d'une première méthode de résolution pour le problème expliqué dans la section 5.
- Mars : algorithmes principaux complétés et vérifiés
- Dimanche 18 mars : rendu de cahier des charges version1
- Mercredi 1 avril : début de l'implémentation d'une première interface
- Mardi 21 avril : Introduction du problème du choix de voiture. Modification des algorithmes pour que ces deux parties s'adaptent
- Lundi 27 avril : rendu du cahier des charges version2, amélioré
- Samedi 11 mai : Interface améliorée. Ajout d'une fonctionnalité qui permet de choisir le sujet du problème d'optimisation
- Mardi 9 juin : rendu des codes, rapport, vidéo

## 8 Développement technique

### PyQt5 et Qt Designer

Nous avons décidé d'implémenter cette interface en utilisant la langage GUI de Python: PyQt5, un module libre qui permet de lier le langage Python avec la bibliothèque Qt. PyQt5 permet ainsi de créer des interfaces graphiques. Ce module nous intéresse pour les raisons suivantes :

- On a appris ce langage pour la première fois dans l'UE IHM de cette année. C'est aussi une bonne opportunité de pratiquer les connaissances acquises.
- Il est relativement simple à utiliser et offre de nombreuses outils d'extensions.
- Il est disponible pour tous les systèmes : Unix, Window et Max OS.
- Son extension Qt Designer permet de gérer directement le code python d'interfaces graphiques.

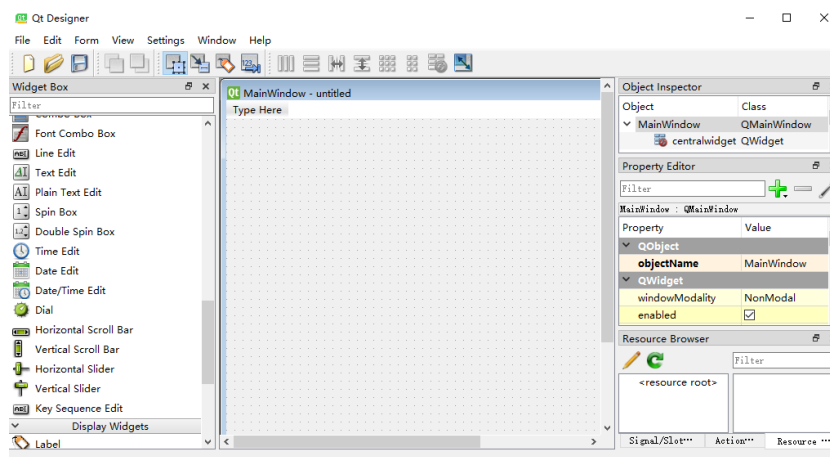


Figure 3: Qt Designer

## 8.1 Signaux et Slot

Des signaux et des slots est le mécanisme de communication de Qt, qui permet une application de type GUI réagit à partir d'évènement.

Un signal est un message envoyé par un widget lorsqu'un évènement se produit. Un slot est la fonction qui sera appelée lorsqu'un évènement s'est produit.

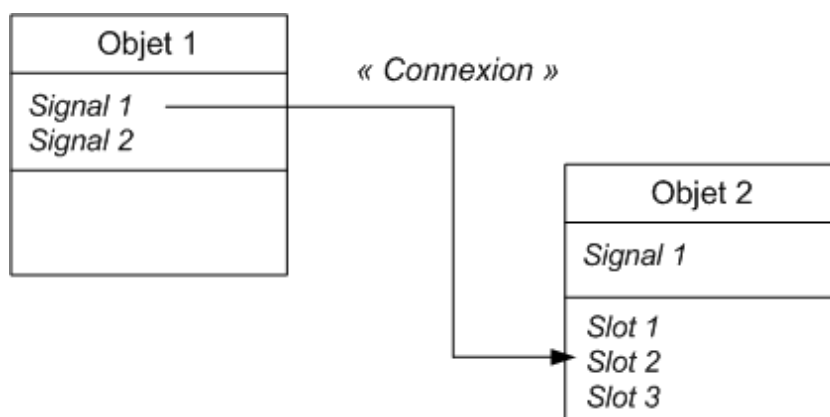


Figure 4: Signaux et slots

On utilise la méthode statitique *connect()* pour connecter le signal et le slot. On peut aussi créer des signaux avec des paramètres qui permettent les échanges entre plusieurs pages.

## 9 Pages et fonctionnalités du produit

Les sections suivantes détaillent les pages et les différentes fonctionnalités du produit final pour la base de données *voiture*.

## 9.1 Le modèle mathématique pour les voitures

La base de données des voitures contient 5 critères: le coût, l'accél, la reprise, le frein, et la tenue de route. On voudrais minimiser les trois premiers et maximiser les deux derniers en même temps. Pour ce genre de problème qui comprend les critères à maximiser et celui à minimiser, on inverse des signes de l'expression à maximiser dans le prétraitement et considère l'objectif est à minimiser la fonction d'agrégation de ce problème.

## 9.2 Landing page

La Figure 2 présente la landing page de notre interface. Cette page remplit le rôle d'introduire et d'expliquer le but de proposer une solution optimale personnalisée. Elle dispose d'un comboBox qui permet à l'utilisateur de choisir avec lequel titre il veut déterminer ses préférences. Le bouton "commencer" et "quitter" permet de commencer le test ou de fermer l'interface.

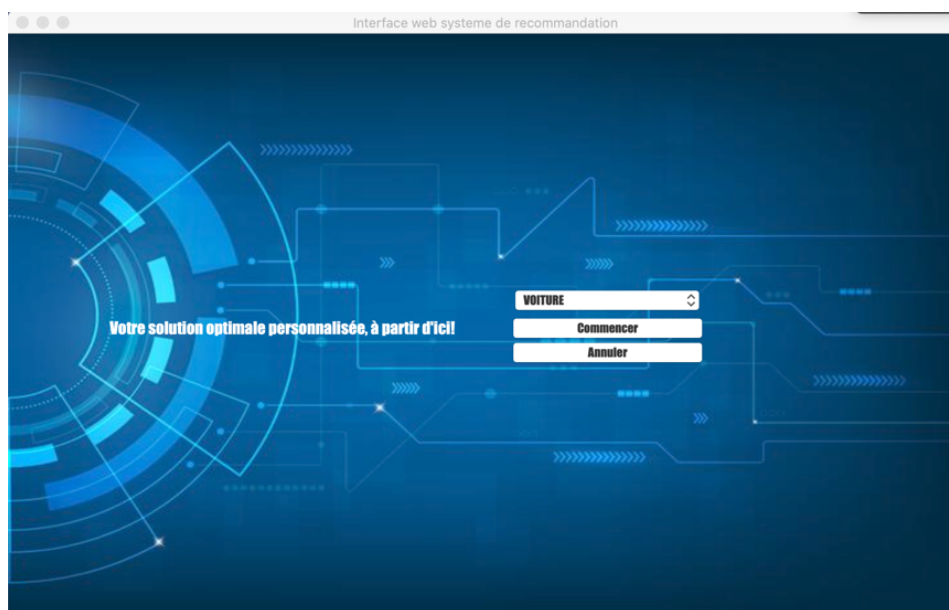


Figure 5: Extrait de landing page

## 9.3 Quiz time

En cliquant sur le bouton "commencer" l'utilisateur commence le quiz.

Comme montré dans la figure suivante, la page quiz est composée de 3 parties : l'alternative qui procède le minimax regret et son pire adversaire, les critères détaillés des deux alternatives et le regret de ce tour est égale à 2.80. Au départ, notre programme linéaire n'a aucune contrainte et il propose voiture *a06* et *a07* pour que le décideur fasse son choix.

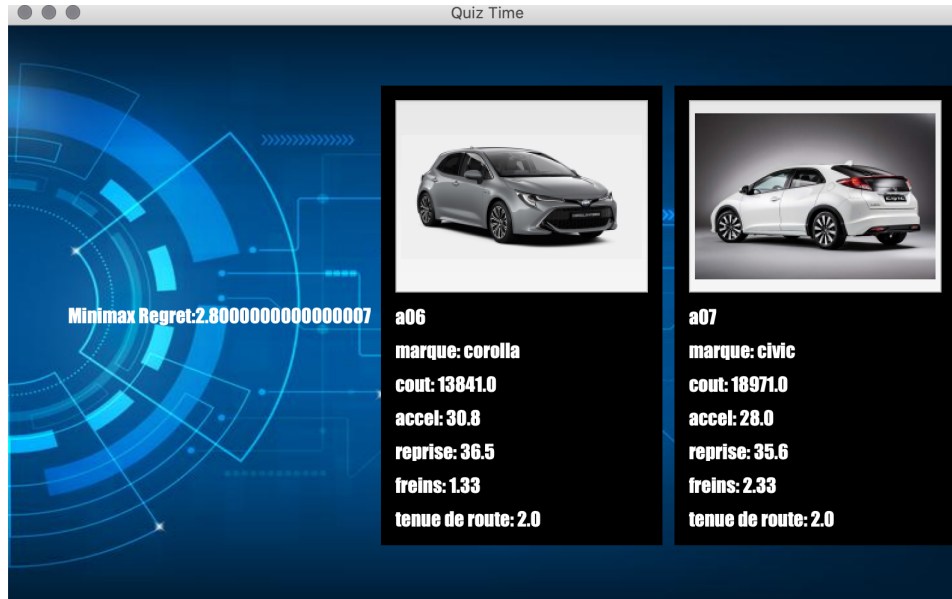


Figure 6: Extrait de page quiz

Supposons-nous le décideur préfère  $a06$  par rapport à  $a07$ . Il donc clique sur l'image de  $a06$  et cette action ajoute une contrainte que  $a06$  est mieux que  $a07$ . Comme on a montré dans l'exemple(section5), la matrice des PMRs vont être recalculée. Et l'alternative de minimax regret et son pire adversaire vont être mises à jour, ici c'est voiture  $a06$  et  $a10$ .

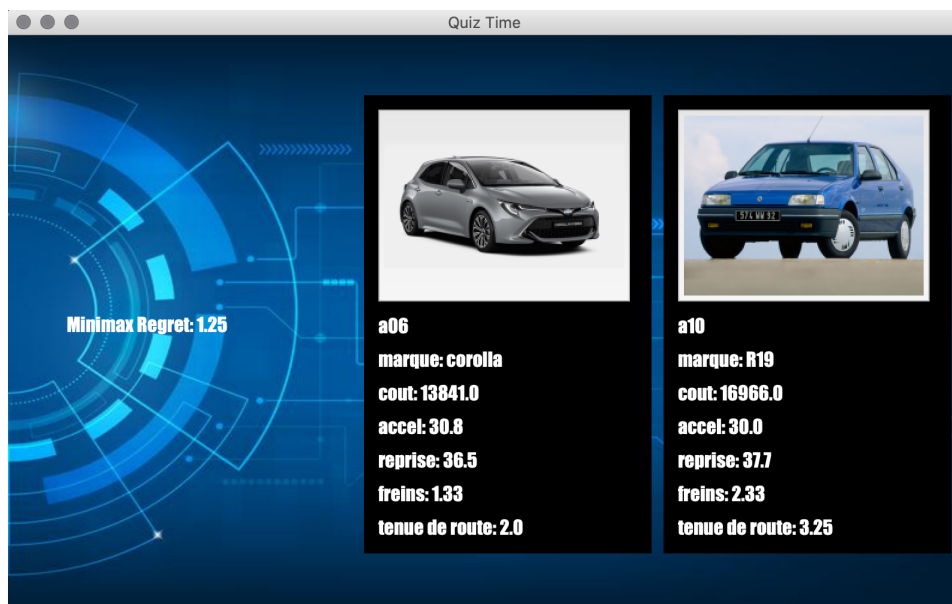


Figure 7: Extrait de page quiz2

A chaque tour, nous calculons d'abord la matrice des PMRs à l'aide du solveur de programme linéaire *Gurobi* (voir annexe1), après nous déterminons une solution de minimax regret et son pire adversaire. L'interface affiche les données détaillées pour les deux solutions

et demande à l'utilisateur de choisir. Chaque choix fait par l'utilisateur ajoute une nouvelle contrainte dans notre programme linéaire, et permet la détermination d'une nouvelle matrice des PMRs pour le prochain tour. D'après la démonstration dans la partie mathématique, on sait que ce regret va diminuer jusqu'à 0 au fur et à mesure des questions et le résultat du test démontre aussi cette théorie.

```
Using license file /Users/wty123/gurobi.lic
Academic license - for non-commercial use only
2.8000000000000007
1.25
1.0513944921943557
0.4645831072701908
0.313501066003951
0.2695192038127271
0.0
quiz end: best car for you is: 3
```

Figure 8: L’affichage des regrets dans le terminal

### 9.4 Solution optimale personnalisée

Une fois le test fini, l'utilisateur peut accéder à sa solution optimale personnalisée. En haut, on précise le nombre des comparaisons qu'il a été faite, c'est à dire le nombre des questions répondues. Parfois un choix peut éliminer plusieurs alternatives. Le bouton "recommencer" permet de retourner sur la landing page et refaire un nouveau test.



Figure 9: Extrait de solution



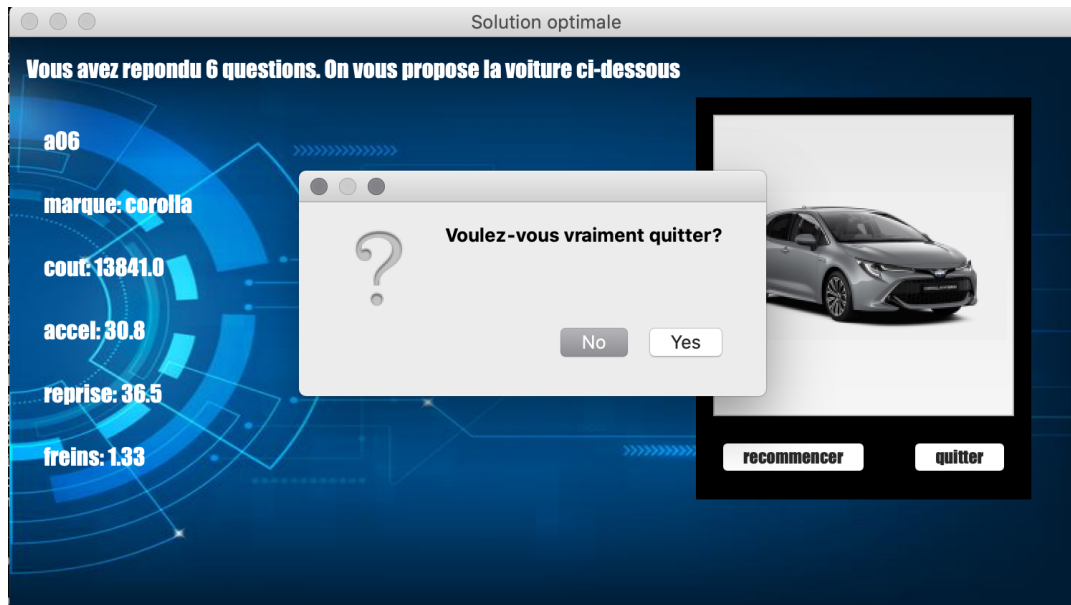


Figure 10: En cliquant le bouton quitter

## 10 Organisation du code

Dans cette partie, nous détaillons comment les codes s'organisent. Depuis la racine du repository, *project/voiture* et *project/maison* enregistrent, la base de données des alternatives, les images des alternatives, les critères.

Les fonctions pour la méthode Minimax Regret sont dans le fichier *fonctions2.py*.

Les codes des pages de notre interface sont enregistrées dans les fichiers *.py* :

- controller.py* : A cause de la méthode de communication *Signal and slot* entre les unités dans Qt, on a besoin de cette classe qui joue un rôle de contrôleur. Elle reçoit des signaux émis par les différentes pages et les traite, décide quelle est la prochaine action: sauter la page ou mettre à jour les informations.

- home4.py*: La landing page.

- mainUI.py*: La page quiz.

- endPage.py*: L'affichage de solution.

- image*: dossier qui enregistre les images des alternatives.

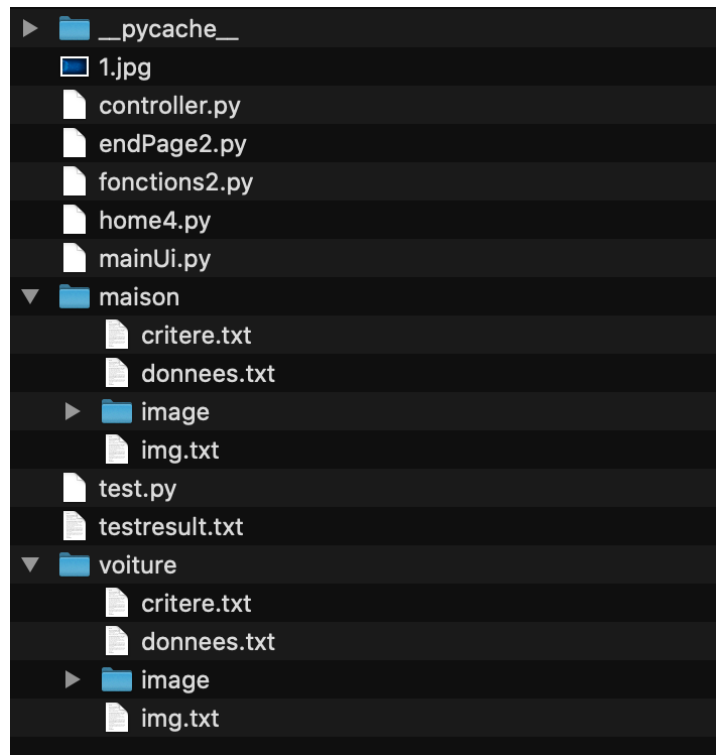


Figure 11: Arborescence des dossiers

## Conclusion

L'implémentation finale du produit répond à la demande initiale du projet: donner l'aide interactive pour ce qui confronté aux problèmes d'optimisation multi-critère. Le développement de cette interface nous permet d'avoir un nouveau regard sur les problématiques qu'on a discutés dans le UE **Decision et Jeux**, de mettre en pratique et de mieux comprendre des connaissances qu'on a acquis pour le UE **Interaction Homme-Machine**. Le projet est l'occasion d'appliquer et tester l'algorithme **Minimax Regret pour élicitation des préférences** sur une interface pour voir s'il résoud bien le problème d'optimisation multi-critères et s'il est le propre algorithme pour une interface interactive.

Ce projet reste encore de nombreuses extensions et améliorations par manque du temps et le problème technique posé par le confinement. Nous le citons ci dessous.

- Il serait nécessaire d'enrichir la base de données des alternatives et d'avoir plus de sujets qui couvre ce qui intéressent les utilisateurs.

- Développement pour que l'interface peut aussi être utilisé en ligne.

- De plus, il serait aussi important de réaliser la fonctionnalité de résoudre le problème d'optimisation combinatoire multi-critères sans les alternatives précis données, par rapport les problèmes simplement "multi-critères".

## Part III

# Annexes

## 11 Gurobi Optimizer

### 11.1 Présentation

Gurobi Optimizer est un logiciel commercial de résolution de programmes mathématiques comme la programmation linéaire, la programmation quadratique, etc. Il prend en charge une variété de langages de programmation, y compris Java, C++ et Python. Nous avons utilisé sa version acadademic license pour résoudre le problème d'optimisation dans notre projet, voici un manuel de ses fonctions principale.

### 11.2 Fonctions principales

```
#Importer le paquet
import gurobipy as gp
#Création d'un modèle
m = gp.Model("m")
#Création des variables pour le modèle
x1 = m.addVar(vtype=GRB.CONTINUOUS, "x1")
x2 = m.addVar(vtype=GRB.CONTINUOUS, "x2")
#Fixer la fonction objective
m.setObjective(x1-x2, GRB.MAXIMIZE)
#Construire expression linéaire
expr=LinExpr()
#Résolution
m.optimize()
#Récupérer le résultat
m.objVal
#Supprimer l'affichage des détails du calcul
m.Params.OutPutFlag=0(par default=1)
```

## 12 Qt et Qt Designer

### 12.1 Fonctions principales Qt

```
#Installation
pip install PyQt5
#Importer des paquets
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
#Créer un QMainWindow comme la fenêtre principale
main = QMainWindow()
main.setWindowTitle("Projet")
```

```

main.show()
#Créer un QWidget comme le composant base, appartient à main
widget = QtWidgets.QWidget(main)
main.setCentralWidget(widget)
#Créer un layout vertical pour widget
vlayout = QtWidgets.QVBoxLayout(widget)
widget.setLayout(vlayout)
#Créer un bouton
pushbutton = QtWidgets.QPushButton(widget)
vlayout.addWidget(pushbutton)
#Créer un label
label=QLabel()
label.setText("Notre projet Androïde")
#Mettre à jour les composants
widget.update()

```

## 12.2 QSS et Qt Designer

QSS, l'abréviation pour "Qt Style Sheet", est utilisé pour décrire la présentation d'une interface écrite en Qt. QSS ressemble beaucoup à CSS, le règle est aussi composé par plusieurs couple de secteur et déclaration. Mais il est fatigant d'ajuster les codes sans voir l'affichage d'interface. Qt Designer nous permet de passer les codes, intègre directement dans l'interface graphique. En téléchargeant le fichier ui, on peut le transformer vers la forme .py grâce au paquet puic5:

```

#Installatioin du paquet
pip install pyuic5-tool
#Transformation
pyuic5 -x file.ui -o file.py

```

## References

- [1] Nawal Benabbou, Christophe Gonzales, Patrice Perny, Paolo Viappiani. *Minimax Regret Approaches for Preference Elicitation with Rank-Dependent Aggregators*. In *EURO Journal on Decision Processes*, 2015, 3 (1-2), p.29-64.
- [2] Nawal Benabbou, Cassandre Leroy, Thibaut Lust, Patrice Perny. *Combining Local Search and Elicitation for Multi-Objective Combinatorial Optimization*. Communication présentée à ADT 2019-6th International Conference on Algorithmic Decision Theory, Duham.
- [3] Nawal Benabbou, Patrice Perny. *Incremental weight elicitation for multiobjective state space search*. In *Artificial Intelligence Conference*, 2015, p.1093–10099.
- [4] Nawal Benabbou, Patrice Perny, Paolo Viappiani. *Incremental elicitation of Choquet capacities for multicriteria choice, ranking and sorting problems*. In *Artificial Intelligence* , 2017, p.125-180.
- [5] Craig Boutilier, Relu Patrascu, Pascal Poupart, Dale Schuurmans. *Constraint-based Optimization and Utility Elicitation using the Minimax Decision Criterion*. In *Artificial Intelligence*, 2017, 170(8–9), p.686–713.
- [6] Darius Braziunas and Craig Boutilier. *Minimax Regret-based Elicitation of Generalized Additive Utilities*. In *Artificial Intelligence Conference*, 2007, p.25-32.
- [7] Eric Brochu, Freitas Nando D, and Abhijeet Ghosh. *Active preference learning with discrete choice data*. In *Internatioinal Neural Information Processing Systems Conference*, 2007,p.409-416.
- [8] B. Chen, J. Wang, L. Wang, Y. He and Z. Wang. *Robust Optimization for Transmission Expansion Planning: Minimax Cost vs. Minimax Regret*. In *IEEE Transaction on Power Systems*, 2014, p.3069-3077.