Université de Nantes
Master 2 parcours "Optimisation en Recherche Opérationnelle (ORO)"
Année académique 2019-2020

Rapport de Projet

Challenge ROADEF 2005 - Car Sequencing Problem

An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem

Andriamiseza.S, Gontier.A, Humbert-Ropers.M, Latif.M

Résumé

Durant ce cours de Métaheuristiques multiobjectif, nous avons étudié le problème du Car Sequencing proposé par l'entreprise Renault lors du challenge ROADEF 2005.
Dans le cadre de ce challenge, plusieurs articles ont été présentés, développant chacun une ou plusieurs méthodes de résolution pour ce problème. Nous nous sommes intéressés au modèle proposé par Prandtstetter et Raidl[2]. Nous nous sommes basés sur ce modèle pour bâtir une approche exacte multiobjectif.
Notre objectif est donc de reproduire le modèle proposé dans cet article et de mettre en place des solutions algorithmiques pour résoudre les instances proposées par l'entreprise Renault et ceci en utilisant des solveurs existants *e.g.* Gurobi ou Cplex.

Table des matières

1 Modèle exact de de Prandtstetter et Raidl.

Pour cette présentation, nous supposons que le lecteur aura, au préalable, pris connaissance du contexte général du challenge et des contraintes liées à l'ordonnancement du film de production. Dans le cas contraire, nous vous invitons à prendre connaissance des détails du challenge en lisant le document proposé par Solnon, Cung, Nguyen et Artigues[3].

Dans cette partie, nous allons expliciter le modèle exact proposé par les auteurs en détaillant les différentes données du problème, variables de décisions et contraintes.

1.1 Données du problème et notations

Nous présentons ci-dessous les différentes notations du modèle :

— C : l'ensemble des composants indicés par c.

— F : l'ensemble des couleurs indicées par f et tel que $F \subseteq C$ *i.e.* les couleurs sont considérées comme un sous ensemble de composants.

— K : l'ensemble des configurations indicées par k

$$K = \{k : K \subseteq C, |k \cap F| = 1\}$$

La relation ci-dessus précise qu'une configuration peut être constituée d'un sous ensemble de composants $c \in C$ mais ne possède qu'une seule couleur.

— $a_{ck} \in \mathbb{B}$ définie telle que pour le jour J

$$a_{ck} = \begin{cases} 1 & \text{si la configuration } k \text{ contient le composant } c \\ 0 & \text{sinon} \end{cases} \quad \forall c \in C \;\; \forall k \in K$$

— $s \in \mathbb{N}$: nombre de voitures possédant une même couleur $f \in F$ autorisées à être ordonnancées consécutivement *aka* paint batch limit ou taille maximale d'une rafale de peinture.

— $m_c \in \mathbb{N}$: taille de la fenêtre glissante pour le composant $\forall c \in C$ - Coefficient P dans le ratio N/P

— $l_c \in \mathbb{N}$: Quota de composant c autorisé durant m_c $\forall c \in C$ - Coefficient N dans le ratio N/P

— $\delta_k \in \mathbb{N}$: demande en voiture de configuration k $\forall \in K$

$$\delta_k = |x_i : x_i = k| \;\forall k \in K$$

— $n \in \mathbb{N}$: nombre de voitures à inclure dans le film de production du jour J.

$$n = \sum_{k \in K} \delta_k$$

— $d_c \in \mathbb{N}$: demande en composant c $\forall c \in C$

$$d_c = \sum_{k \in K} a_{ck}\delta_k \;\forall c \in C$$

— $\gamma_f \in \mathbb{N}$: pénalité associée à un changement de peinture $\forall f \in F$

— $\gamma_c \in \mathbb{N}$: pénalité associée à la violation de la contrainte de ratio N/P $\forall c \in C \setminus F$

— $e_{ci} \in \mathbb{B}$ définie telle que pour le jour $J-1$

$$e_{ci} = \begin{cases} 1 & \text{si la voiture en position } i \text{ nécessite le composant } c \\ 0 & \text{sinon} \end{cases} \quad \forall c \in C, \forall i \in [\![1, m_c - 1]\!]$$

— $e_{fi} \in \mathbb{B}$ définie telle que pour le jour $J-1$

$$e_{fi} = \begin{cases} 1 & \text{si la voiture en position } i \text{ nécessite la couleur } f \\ 0 & \text{sinon} \end{cases} \quad \forall f \in F, \forall i \in [\![1, m_c - 1]\!]$$

1.2 Variables de décisions

La modélisation du problème repose sur quatre variables de décisions présentées ci-dessous ; Pour un film de production de longueur n et chaque position $i \in [\![1, n]\!]$ dans ce même film :

— $b_{ci} \in \mathbb{B}\ \forall c \in C, \forall i \in [\![1, n]\!]$:

$$b_{ci} = \begin{cases} 1 & \text{si le composant } c \text{ est installé dans le véhicule en position } i \text{ du film} \\ 0 & \text{sinon} \end{cases}$$

Note : Remarquons que cette variable de décision est définie pour l'ensemble des composants C incluant également les peintures.

— $p_{k,i} \in \mathbb{B}\ \forall k \in K, \forall i \in [\![1, n]\!]$:

$$p_{ki} = \begin{cases} 1 & \text{si le véhicule produit en position } i \text{ correspondent à la configuration } k \\ 0 & \text{sinon} \end{cases}$$

— $w_{fi} \in \mathbb{B}\ \forall f \in F, \forall i \in [\![1, n]\!]$:

$$w_{fi} = \begin{cases} 1 & \text{si un changement de couleur } f \text{ est opéré à la position } i \text{ du film} \\ 0 & \text{sinon} \end{cases}$$

— $g_{ci} \in \mathbb{N}\ \forall c \in C \setminus F, \forall i \in [\![1, n]\!]$, le nombre de violation(s) de la contrainte de ratio pour le composant c en position i dans le film.

1.3 Modèle :

Nous présentons ci-dessous le modèle proposé par les auteurs pour la résolution exacte du Car Sequencing Problem

$$\min \sum_{c \in C \setminus F} \gamma_c \times \sum_{i=1}^{n} g_{ci} + \sum_{f \in F} \gamma_f \times \sum_{i=1}^{n} w_{fi} \tag{1}$$

$$\text{s.t} \sum_{f \in F} b_{fi} = 1 \qquad \forall i \in [\![1, n]\!] \tag{2}$$

$$\sum_{i=1}^{n} b_{ci} = d_c \qquad \forall c \in C \tag{3}$$

$$p_{ki} \leq a_{ck} b_{ci} + (1 - a_{ck})(1 - b_{ci}) \qquad \forall k \in K, \forall c \in C, \forall i \in [\![1, n]\!] \tag{4}$$

$$b_{ci} = \sum_{k \in K} a_{ck} p_{ki} \qquad \forall c \in C, \forall i \in [\![1, n]\!] \tag{5}$$

$$\sum_{i=1}^{n} p_{ki} = \delta_k \qquad \forall k \in K \tag{6}$$

$$g_{ci} \geq \sum_{j=1}^{i} b_{cj} + \sum_{j=1}^{m_c - i} e_{cj} - l_c \qquad \forall c \in C \setminus F, \forall i \in [\![1, m_c - 1]\!] \tag{7}$$

$$g_{ci} \geq \sum_{j=i-m_c+1}^{i} b_{cj} - l_c \qquad \forall c \in C \setminus F, \forall i \in [\![m_c, \forall n]\!] \tag{8}$$

$$w_{f1} \geq b_{f1} - e_{f1} \qquad \forall f \in F \tag{9}$$

$$w_{fi} \geq b_{fi} - b_{f(i-1)} \qquad \forall f \in F, \forall i \in [\![2, n]\!] \tag{10}$$

$$\sum_{j=1}^{i} b_{fi} + \sum_{j=1}^{s+1-i} e_{fi} \leq s \qquad \forall f \in F, \forall i \in [\![1, s]\!] \tag{11}$$

$$\sum_{j=i-s}^{i} b_{fj} \leq s \qquad \forall f \in F, \forall i \in [\![s+1, n]\!] \tag{12}$$

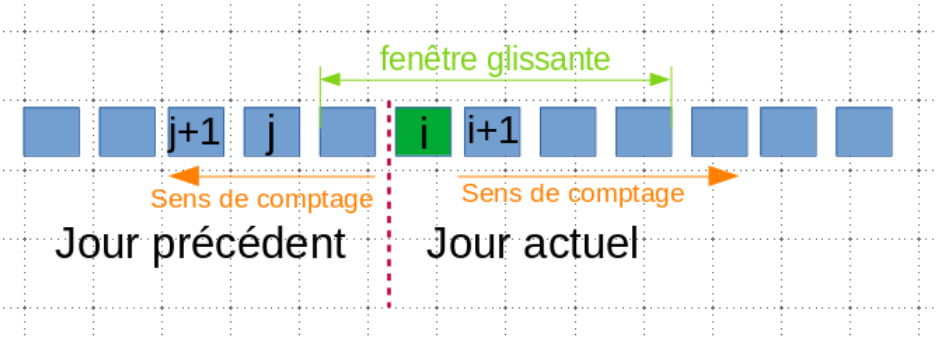
$$g_{ci} \geq 0 \qquad \forall c \in C \setminus F, \forall i \in [\![1, n]\!] \tag{13}$$

$$w_{fi} \in \{0, 1\} \qquad \forall f \in F, \forall i \in [\![1, n]\!] \tag{14}$$

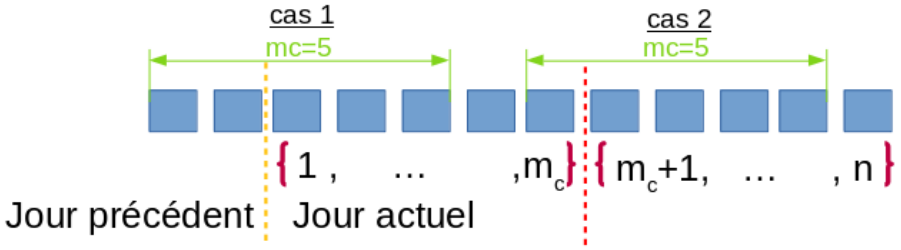
$$b_{ci} \in \{0, 1\} \qquad \forall c \in C, \forall i \in [\![1, n]\!] \tag{15}$$

$$p_{ki} \in \{0, 1\} \qquad \forall k \in K, \forall i \in [\![1, n]\!] \tag{16}$$

Il y a principalement deux facteurs importants qui régissent le problème ; ce sont les composants à placer dans les voitures et la couleur des voitures. Les deux objectifs à minimiser sont donc bien le nombre de contraintes violées à la position i au regard d'un composant c et le nombre de changements d'une couleur f à une position i donnée. La fonction objectif (1) permet de compter le nombre de violations de contraintes de ratio ainsi que les changements de peinture. Les valeur des pénalités γ_c et γ_f sont données par l'instance.
La contrainte (2) nous permet de spécifier que chaque voiture doit être peinte avec une seule et unique couleur. Dans ce modèle, nous dénombrons deux contraintes de *demandes* qui doivent être respectées : la contrainte (3) permettant de respecter le besoin en composant total pour chaque véhicule produit avec le composant $c \in C$ et la contrainte (6) permettant d'assurer la production du bon nombre de véhicules possédant une certaine configuration et ceci conformément au carnet de commande.
La contrainte (4) permet d'assurer la la cohérence entre les configurations et les positions du film. Rappelons que $p_{k,i}$ correspond à une variable binaire indiquant que la présence d'une configuration k en position i. Ainsi, $p_{k,i}$ ne peut valoir 1 que si les composants placés en position i correspondent bien à la configuration k ; Dès lors qu'un composant n'est pas installé à la position i alors $p_{k,i}$ vaudra 0. Si $b_{c,i} = 0$ en soit si c n'appartient à la position i alors $a_{c,k} \times b_{c,i} = 0$ au regard de tous les composants il y en a forcément qui appartiennent à la configuration k ainsi : $\exists c : a_{c,k} = 1$ et donc $(1 - a_{c,k})(1 - b_{c,i}) = (1 - 1)(1 - 1) = 0$. On a nécessairement $p_{k,i} \leq 0 \times 0$ et donc dés lors qu'un composant c n'apparaît pas à la position i, alors toute configuration k où c est présent est forcé à 0.
Comme pour les composants, nous devons également nous assurer que pour une configuration k installée en position i, tous les composants c associés à k seront correctement placés à la position i ; ce besoin est traduit par la contrainte (5) du modèle.
La contrainte (7-8) sont relatives au comptage des contraintes de ratio violées.

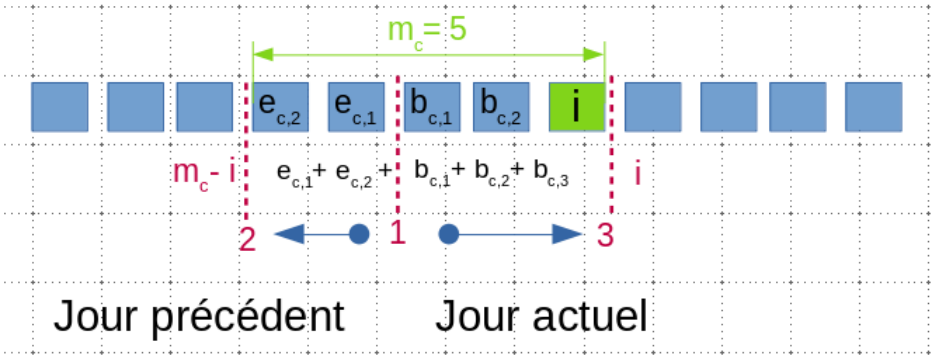


Pour traduire la contrainte de viol de ratio, il convient de considérer "parties" dans le film de production :
Cas 1 correspond à l'ensemble des positions comprises entre $[\![1, m_c]\!]$ et ceci pour tout composant $c \in C$
Cas 2 correspond à l'ensemble des positions comprises entre $[\![m_c + 1, n]\!]$ et ceci pour tout composant $c \in C$

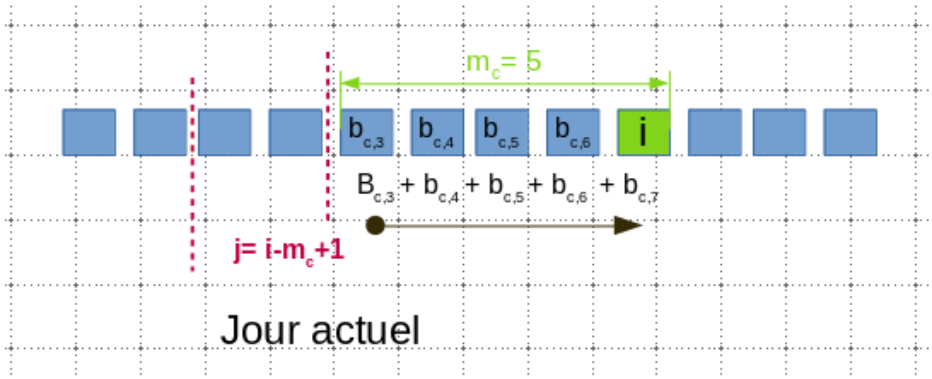


Considérons le cas 1 : rappelons que $g_{c,i}$ est le nombre de violations de contraintes à la position i au regard du composant c. Nous supposerons que l'on compte à partir de la position i. La variable $e_{c,j}$ correspond au composant c à la position j posé lors dans la production de la journée précédente. Nous nous placerons dans le cadre de la fenêtre glissante m_c en considérant la position i. Prenons par exemple les valeurs $m_c = 5$ et $i = 3$. En considérant la production de la veille, nous allons parcourir les positions allant de 1 à $m_c - i = 5 - 3 = 2$ et pour la journée courante, nous allons parcourir les positions comprises entre 1 et 3. Pour chaque composant

$c \in C$, la présence de ce dernier dans le film aux positions étudiées est donnée par la somme suivante : $e_{c,1} + e_{c,2} + b_{c,1} + b_{c,2} + b_{c,3}$



Étant donné que l_c représente le nombre de voitures autorisées à posséder le composant c dans la fenêtre considérée, nous pouvons supposer qu'il y a 5 composants c présents dans la fenêtre mais que l_c est égale à 3. Nous aurons $g_{c,i} = 5 - 3 = 2$ 2 violations sur le composant c à cette position i donnée.
Considérons à présent le cas 2 : nous nous placerons dans le cadre de la fenêtre glissante m_c par rapport à une position i avec $i > m_c$, cela nous assure alors de n'avoir jamais à considérer la production du jour précédent.



Sans perte de généralité et considérant les explications du cas 1, nous obtenons la contrainte (8) telle que

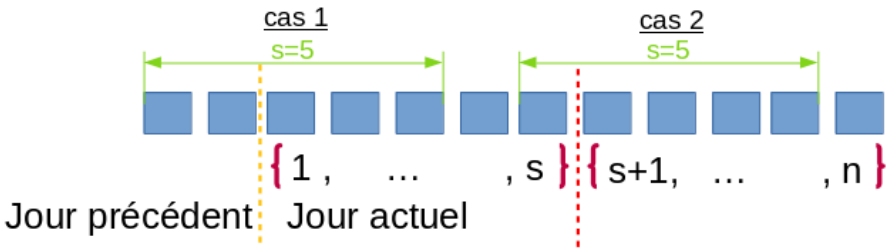
$$g_{ci} \geq \sum_{j=i-m_c+1}^{i} b_{cj} - l_c \; \forall c \in C \setminus F, \forall i \in [\![m_c, n]\!]$$

Les contraintes liées au comptage du nombre de changements de peinture sont données par les inéquations (9-10) du modèle. Considérons l'exemple suivant : Nous nous plaçons en position $i = 1$ du film et nous nous assurons qu'il existe un changement de peinture par rapport à la dernière voiture produite la veille ce qui se modélise avec nos variable par $w_{f,1} = 1$; pour une couleur $f \in F$ choisie arbitrairement, nous pouvons définir les quatre situations possibles :

Cas	$b_{f,1}$	$e_{f,1}$	$b_{f,1} - e_{f,1}$	Changement de couleur
1	1	1	0	Non
2	0	0	0	Non
3	1	0	1	Oui
4	0	1	-1	Oui

Dans les cas 1 et 2 présentés ci-dessus, il n'y a pas de changement de peinture. cette absence est vérifiée par la valeur de la variable $w_{f,1} \geq 0$. Le cas 3 présente un changement de peinture et nous retrouvons bien une

valeur de la variable $w_{f,1} = 1$. Nous notons également le cas 4 où nous constatons un changement de peinture mais dans ce cas, $w_{f,1} \geq -1$; cependant, cela implique qu'il existe une couleur f' différente de f tel que l'on ait $b_{f',1} = 1$ et $e_{f',1} = 0$, nous pouvons alors nous ramener à un changement de couleur tel que $w_{f',1} \geq 1$. Le cas présenté est généralisable pour chaque positions dans le film de production.
Enfin, les contraintes (11-12) modélisent la limite de rafale de peinture ; cette contrainte contraint le nombre maximal de voiture peintes de la même couleur. De la même manière que pour le comptage des violations de contraintes de ratio, nous allons nous reposer sur la fenêtre glissante en séparant le film de production en deux intervalles ; le premier constitué des positions allant de $[\![1, s]\!]$ intégrant la production du jour précédent et le second intervalle composé des positions allant de $[\![s + 1, n]\!]$ qui, quant à lui, ne compte que la production du jour concerné.



Enfin, les contraintes (13-16) assurent l'intégrité des variables de notre modèle.

1.4 Construction des paramètres du modèle à partir des instances de Renault

L'intégration des données fournies fût assez délicate. En effet, nous avons dû retrouver le lien qui permet de construire le modèle à partir des instances car celui-ci n'était pas donné dans l'article. Nous avons donc développé la structure de données *RawData* qui stockait les données brutes :

```
# Données brute sans pré-traitement issues des instances
mutable struct RawData
    limitation::Int
    # Attribut(s) issus du fichier optimization objectives.txt
    nbObj::Int
    sortedObj::Vector{String}
    # Attribut(s) issus du fichier ratio.txt
    ratioString::Vector{String}
    N::Vector{Int}
    P::Vector{Int}
    prio::Vector{Int}
    # Attribut(s) issus du fichier vehicles.txt
    colorJ0::Vector{Int}
    matriceHLJ0::Matrix{Int}
    colorJ1::Vector{Int}
    matriceHLJ1::Matrix{Int}
end
```

Dès lors que nous avons obtenu ces données brutes, nous avons pu opérer des pré-traitements sur les données parsées.

Principe de boolearisation des couleurs et extraction des configurations

Dans les instances fournies par Renault, les couleurs des véhicules sont codées par des entiers alors que les matrices K et $a_{c,k}$ sont booléennes. Nous avons donc choisi d'encoder les couleurs sous forme de vecteurs binaires et de concaténer ces vecteurs aux configurations déjà parsées. Prenons par exemple un carnet de commande

composé de 4 voitures, 2 composants et 2 couleurs. Après application des opérateurs de boolearisation et de concaténation, nous obtenons la représentation du carnet de commande suivante :

	Composants		Couleurs	
	HPRC	LPRC	vert	bleu
Voiture 1	0	1	1	0
Voiture 2	1	1	1	0
Voiture 3	0	1	0	1
Voiture 4	1	1	1	0

On peut remarquer que les voitures 2 et 4 possèdent les même configurations, nous pouvons donc les agréger et on peut en extraire une matrice dont chaque ligne sera une configuration unique k qui sera comme suit.

$$K = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

A partir de la matrice des configurations K, nous pouvons obtenir aisément la matrice $a_{c,k}$ par application de l'opérateur transposé :

$$a_{c,k} = K^\top = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Ordre des objectifs

L'ordre des fonctions objectifs varie selon les instances du problème. Il a donc été nécessaire de pouvoir adapter en conséquence les valeurs de γ_f et γ_c. A partir des chaînes de caractères lu dans les fichiers d'instances et stockés dans la structure de données brutes, nous déterminons les pondérations de γ_f et γ_c pour la résolution de la somme pondérée.

Ajustements des indices

Lors de l'implémentation des contraintes de paint batch limit, nous nous sommes rendu compte que ces dernières nécessitaient plus d'informations concernant les productions de la journée précédente. Nous avons donc dû recalculer l'étendue de la taille de la rafale peinture m_l sur le jour précédent en fonction des données réelles que nous avions à notre disposition. Nous avons dû réaliser le même ajustement pour les tailles des fenêtres pour chaque composant noté m_c.

Pour conclure cette section, nous présentons ci-dessous la structure de données que nous avons utilisée pour la suite de nos implémentations. Les fonctions utiles pour comprendre toutes les transformations réalisées à partir des données brutes sont implémentées dans le fichier *src/tools.jl*.

```
mutable struct Instance
    # Attribut(s) optimization objectives
    nbObj::Int
    sObj::Vector{String}
    # Nombre de voitures à produire (mickaels?)
    nbPos::Int
    # paint bash limit (longueur maximale de rafale de peinture)
    s::Int
    # Nombre de composants excluant les peintures
    nbCmF::Int
    # Nombre composants totaux (incluant les peintures)
    nbC::Int
```

```
    # Nombre de configurations différentes
    nbK::Int
    # Nombre de voitures pour chaque configuration k
    delta::Vector{Int}
    # Demande en composant c pour chaque composant c
    d::Vector{Int}
    # matrice décomposant l'appartenance des composants aux configurations
    a::Matrix{Int} journée J1
    # équilvalent au N du ratio : quota à respecter pour le composant c
    l::Vector{Int}
    # équilvalent au P du ratio : taille de la fenêtre glissante pour le composant c
    m::Vector{Int}
    # vecteur traduisant les priorités 1 si prio High 0 sinon
    h::Vector{Int}
    # matrice décomposant l'appartenance des composants aux configurations journée J0
    e::Matrix{Int}
    # Taille de la paint batch limite (issue du post-traitement)
    ml::Int
    # Taille de la fenêtre pour chaque composant (issue du post-traitement)
    mc::Vector{Int}
end
```

2 Démarche du projet

Suite à l'étude du modèle et des données, nous avons décidé d'utiliser ce dernier comme base pour le développement de méthode de résolution multi-objectif. En effet, depuis le challenge, de nouvelles méthodes de résolution tri-objectif ont vu le jour comme par exemple celle Kirlik et Sayin proposée en 2014. Afin de mettre en oeuvre cette méthode nous avons d'abord mis en place un modèle JuMP (Julia Math Programming) reprennant les instances parsées données par Renault lors du challenge ROADEF 2005.
Tout d'abord, nous avons réalisés des expérimentations sur les problèmes mono-objectifs disponibles, soit lexicographique soit par somme pondérée des objectifs. Nous avons ensuite amélioré leur temps de résolution pour permettre la réutilisation de ces modèles dans le cadre, dans un premier temps, de l'ε-contrainte puis dans un second temps, pour la méthode de résolution tri-objectif de Kirlik et Sayin.
Ainsi, nous allons d'abord présenter le principe de ces méthodes multi-objectifs pour ensuite introduire les résultats, en expliquant les évolutions et améliorations apportées au long du développement de la méthode.

3 ε-contrainte

3.1 Principe

La méthode ε-contrainte bi-objectif permet une résolution exacte et l'énumération de l'ensemble des points non dominés.
L'idée est de transformer le problème bi-objectif en un problème mono-objectif en passant un objectif en contrainte. Ainsi, nous pouvons contraindre les valeurs de la fonction objectif en contrainte et procéder à des résolutions mono-objectif afin d'obtenir l'ensemble des points non dominés.
L'inconvénient de la méthode est que si l'on arrête en cours de route l'ε-contrainte, les points non dominés fournis ne sont pas uniformément repartis dans les espaces des objectifs.

3.2 Déroulement

Soit un problème bi-objectif P. Soit X l'ensemble des solutions admissibles, z_1 et z_2 deux fonctions objectifs.

$$\begin{array}{lll} \text{Min} & z_1(x) & \\ \text{Min} & z_2(x) & \\ \text{s.t.} & x \in & X \end{array}$$

où X est la région admissible du problème .

La méthode ε-contrainte se base sur la résolution itérative de PL mono-objectif. Pour ce faire, une fonction objectif est mise en contrainte. On choisit ici arbitrairement de mettre z_2 en contrainte. Elle est contrainte par une constante ε, d'abord fixé à ∞. L'idée étant de partir de l'optimum lexicographique pour la fonction z_1 et, ensuite, de tapisser l'espace des solutions. ε est diminué progressivement pour trouver chaque point non dominé, jusqu'à ce que l'on trouve l'optimum lexicographique pour la fonction z_2 ; c'est-à-dire jusqu'à ce qu'il n'y ait plus de points réalisables en-dessous d'un certain ε.

Nous avons donc le problème mono-objectif suivant :

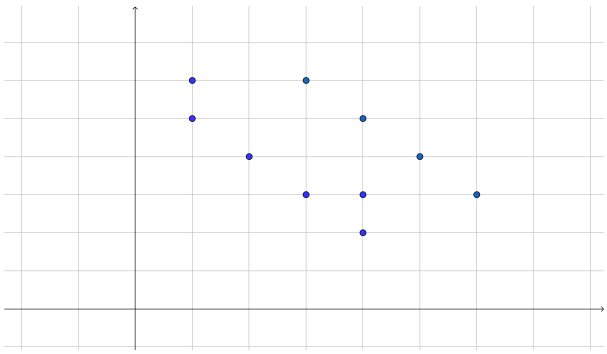
$$\begin{array}{lll} \text{Min} & z_1(x) & \\ \text{s.t.} & x \in & X \\ & z_2(x) \leq & \epsilon_k \end{array}$$

avec ε_k la valeur à la k-ième résolution.

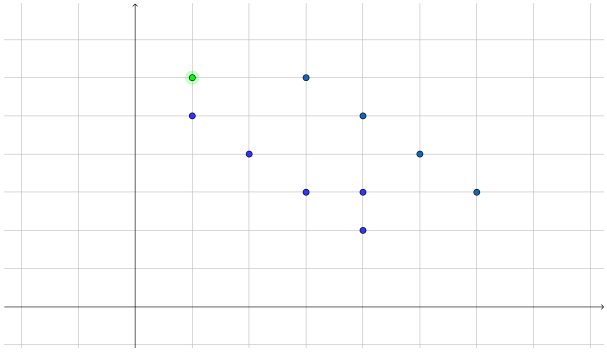
L'ε entre deux itérations se met à jour de la manière suivante :
Notons $\hat{x}_k$ la solution optimale fournie par la résolution précédente. D'ailleurs, cette solution est au moins faiblement efficace. Comme nous sommes dans le cadre d'un problème combinatoire multi-objectif avec des coûts entiers, ε est fixé à $\varepsilon_{k+1} = z_2(\hat{x}_k) - 1$, afin de trouver une autre solution faiblement efficace.
Si une solution x' est faiblement efficace, la résolution suivante permet de trouver une solution qui la domine, et donc x' est dominée. Toutes les solutions dominées sont donc filtrées au fur et à mesure. Nous présentons ce déroulement dans les graphiques de la section suivante.

3.3 Intuition graphique

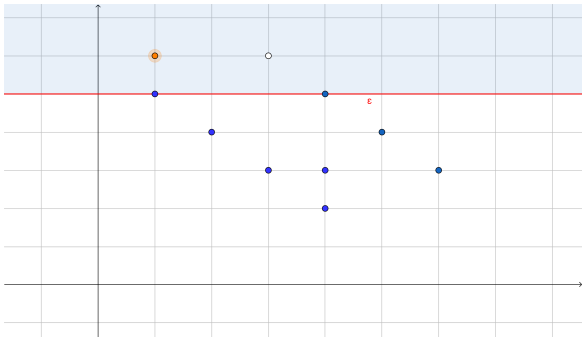
Voici notre ensemble de points dans l'espace des objectifs.



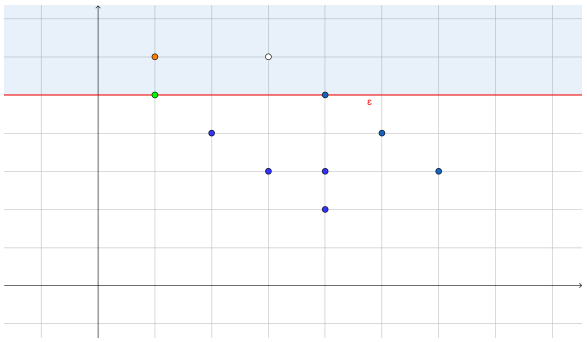
Notre première résolution au regard de l'objectif z_1 nous donne le point en vert.



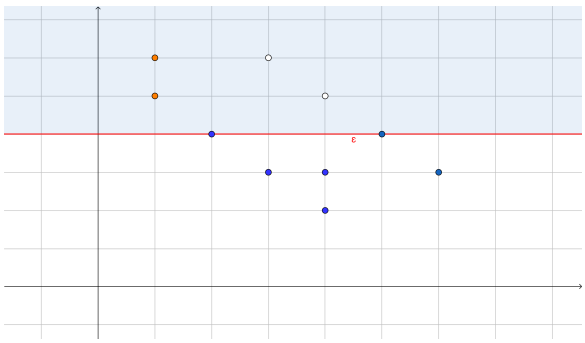
Le nouveau problème est construit à l'aide de la solution précédente. Les points non dominés sont sauvegardés dans une liste. Les points non dominés sont en orange sur le graphique.



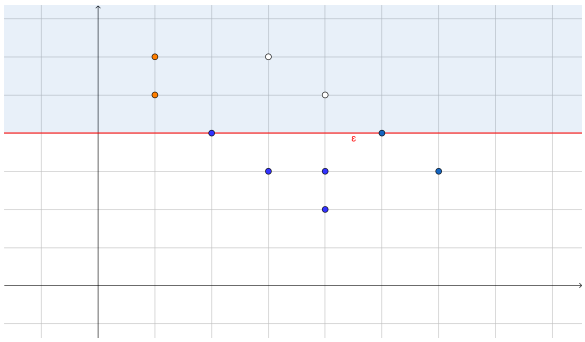
Un nouveau point non dominé est trouvé en minimisant sur z_1 ici en vert.

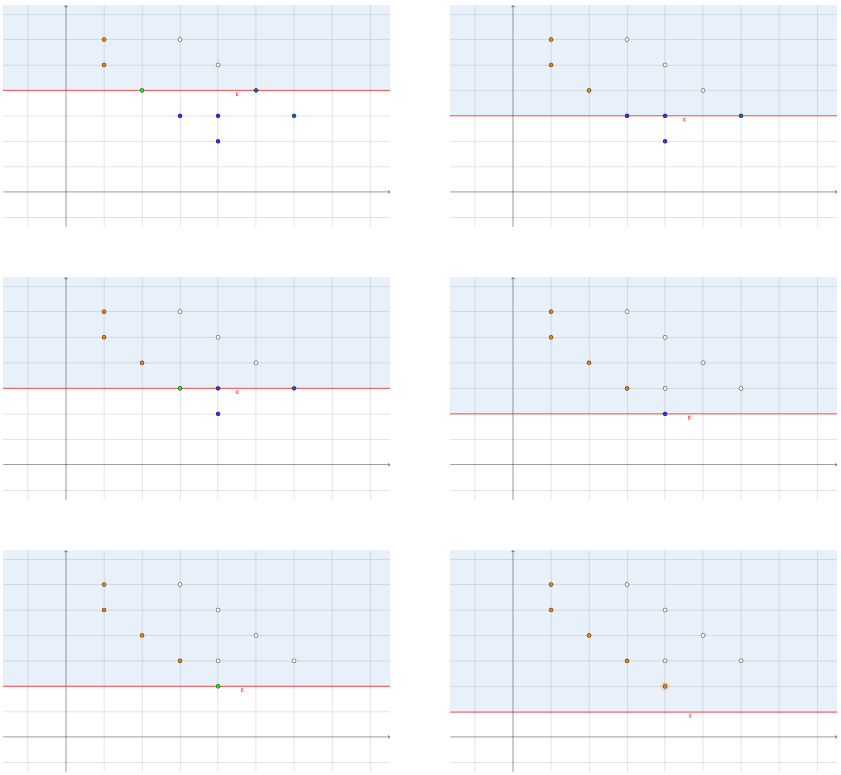


On construit encore un nouveau PL à l'aide de la solution verte précédente.

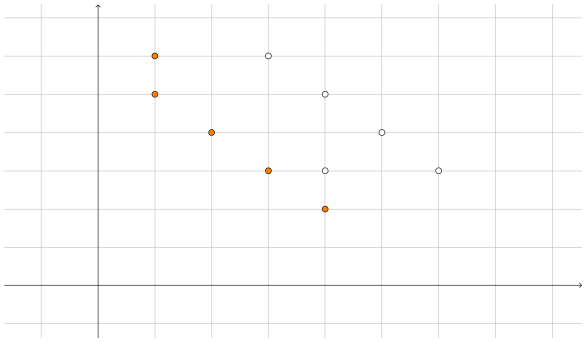


Les itérations se poursuivent tant que possible sans oublier de sauvegarde les points non dominés.





Tout l'ensemble des points non dominés est obtenu (en orange)



4 Kirlik et Sayin

La méthode de Kirlik et Sayin[1] est une méthode exacte pour la résolution de problème d'optimisation combinatoire multi-objectif (i.e. permettant de passer à 3 objectifs et plus). La description donnée ci-dessous se limite au cas tri-objectif, qui nous intéresse dans ce cas d'étude.

$$\begin{array}{lll} \min & z_1(x) & \\ \min & z_2(x) & \text{(P)} \\ \min & z_3(x) & \\ \text{s.t} & x \in X & \end{array}$$

où X est la région admissible du problème.

La méthode de Kirlik et Sayin repose sur la méthode ε-contrainte. Cette méthode consiste à ne garder qu'une fonction objectif et à transférer les autres fonctions objectifs dans les contraintes avec une borne supérieure sur la valeur de ces fonctions objectifs. Par exemple, si on transfère les fonctions objectifs z_2 et z_3 dans les contraintes, on obtient :

$$\begin{array}{lll} \min & z_1(x) & \\ \text{s.t} & z_2(x) \leq \varepsilon_2 & \text{(P1)} \\ & z_3(x) \leq \varepsilon_3 & \\ & x \in X & \end{array}$$

Où X est la région admissible du problème, et $\varepsilon_2, \varepsilon_3 \in \mathbb{R}$ sont des paramètres à fixer. Les propriétés de $P_1(\varepsilon_2, \varepsilon_3)$ sont les suivantes :

— Si $P_1(\varepsilon_2, \varepsilon_3)$ est réalisable alors sa solution optimale est au moins faiblement efficace pour le problème tri-objectif initial (P).

— Pour chaque solution efficace $x \in X_E$, il existe des paramètres $(\varepsilon_2, \varepsilon_3)$ tels que x est une solution optimale de $P_1(\varepsilon_2, \varepsilon_3)$.

Le principe des méthodes ε-contraintes avec pas adaptatif (dont fait partie la méthode de Kirlik et Sayin) est de fixer les paramètres $(\varepsilon_2, \varepsilon_3)$ en fonction des résolutions déjà effectuées de manière à chercher de nouveaux points non-dominés, ou de conclure qu'il n'y en a plus à trouver. Ces méthodes déterminent un ensemble complet minimum, c'est-à-dire qu'elles garantissent l'obtention d'au moins une solution efficace pour chaque point non-dominé.

Afin d'éviter de générer et stocker des solutions faiblement efficaces (qui seront remplacées ultérieurement par des solutions efficaces), il est souvent proposé de résoudre un second problème à la suite du problème $P_1(\varepsilon_2, \varepsilon_3)$. Si x^1 est une solution optimale de $P_1(\varepsilon_2, \varepsilon_3)$ et $y^1 = z(x^1)$. Le problème suivant peut alors être résolu.

$$\begin{array}{lll} \min & z_2(x) + z_3(x) & \\ \text{s.t} & z_1(x) \leq y^1 & \text{(P2)} \\ & z_2(x) \leq c_2 & \\ & z_3(x) \leq c_3 & \\ & x \in X & \end{array}$$

La résolution de ce problème permet soit de confirmer que x^1 est efficace, soit d'obtenir une solution efficace dominant x^1.

L'espace des paramètres est ici un plan défini par les objectifs z_2 et z_3, seul les valeurs permettant d'obtenir de nouveaux points non dominés nous intéressera. Clairement, le rectangle défini par les projections des point idéal et anti-idéal couvrent bien toutes les valeurs possibles pour les paramètres. Ce rectangle sera découpé progressivement en utilisant la projection de chaque nouveau point non dominé sur le plan z_2, z_3. Cela augmente bien sûr le nombre de sous-rectangles mais, Kirlik et Sayin s'appuient sur deux propriétés permettant de conclure que certains sous-rectangles ne contiennent les projections d'aucun point non dominé qu'il reste à trouver. Les valeurs ε_2 et ε_3 seront fixés à l'aide des points supérieurs droit des rectangles auxquels seront soustrait $(-1, -1)$ pour éviter de retomber sur des solutions déjà connues.

Définition Dans le plan, soit deux points y^1 et y^2, tel que $y^1 \geq y^2$. Le rectangle $R(y^1, y^2)$ est défini dans l'espace des objectifs par son point inférieur gauche y^1 et supérieur droit y^2.

Par exemple, le rectangle $R(y^I, y^{AI})$ proposé à l'initialisation de l'algorithme est celui définit avec le point idéal, noté y^I, et anti-idéal, noté y^{AI}.

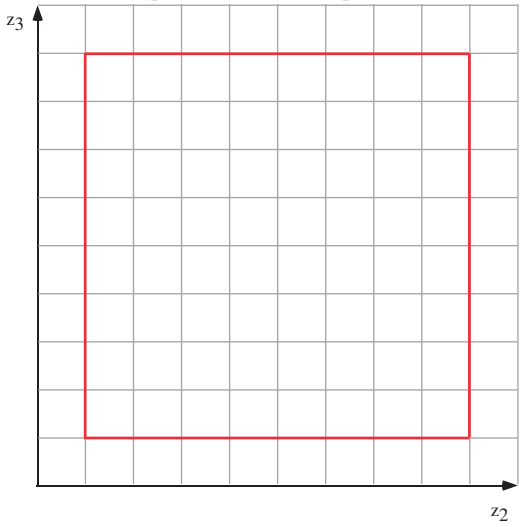
Lemme. *Soit x^* une solution optimale obtenue lors de l'exploration d'un rectangle $R(y^I, u)$, c'est-à-dire obtenue en résolvant un problème ε-contrainte avec $\varepsilon = u - (\varepsilon, \ldots, \varepsilon)$. Alors, il n'existe aucun point dans le rectangle $R(z(x^*), u)$, à part x^*.*

Remarque. Notons aussi que nous sommes dans le cadre d'un problème combinatoire avec des valeurs entières, ainsi il est possible de choisir de poser $\varepsilon = 1$

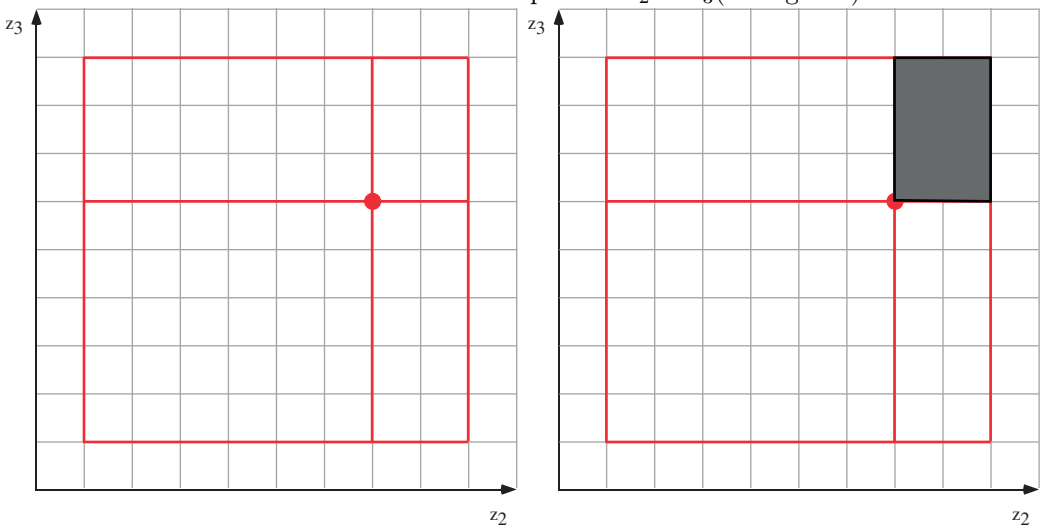
Lemme. *Si un problème ε-contraint avec $\varepsilon = u - (\varepsilon, \ldots, \varepsilon)$ n'a aucune solution réalisable, alors, il n'existe aucune solution admissible dans le rectangle $R(y^I, u)$.*

Pour mieux comprendre le déroulement de la méthode, voici une résolution graphique d'un problème tri objectif :

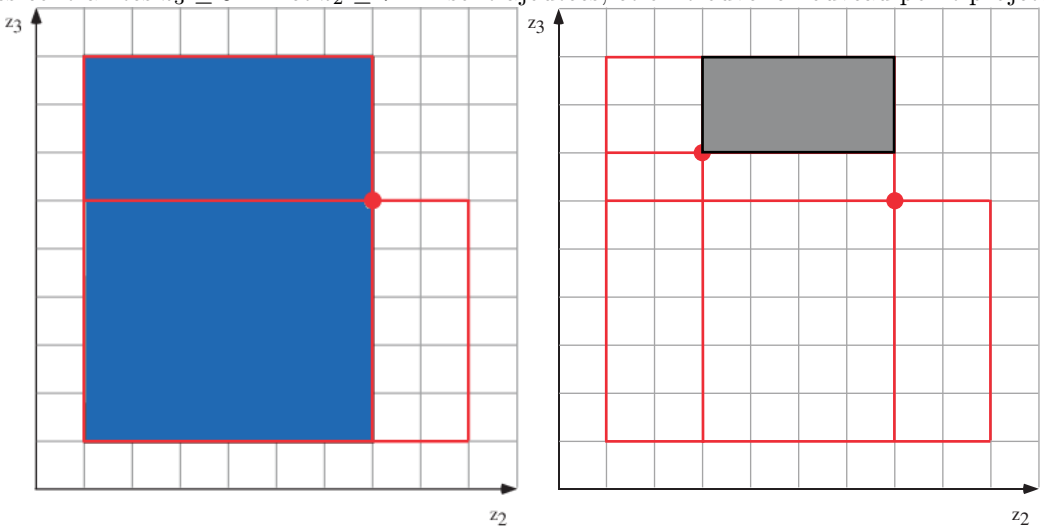
Un premier rectangle est créé avec le point idéal et le point anti-idéal du problème $R(y^I, y^{AI})$.



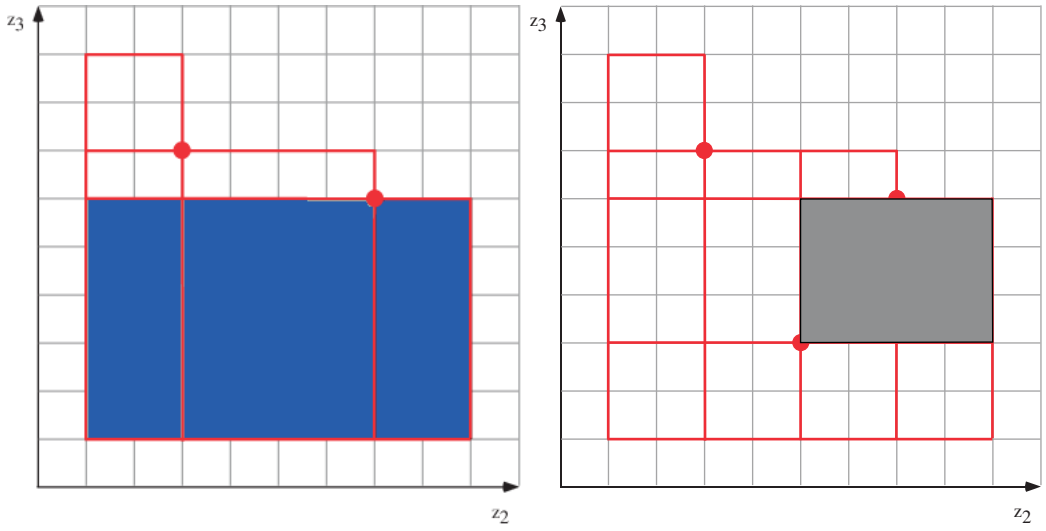
La première résolution nous donne un point de l'ensemble des points non dominés projeté selon z_1 et une zone dominée dans l'espace de z_2 et z_3(zone grisée)



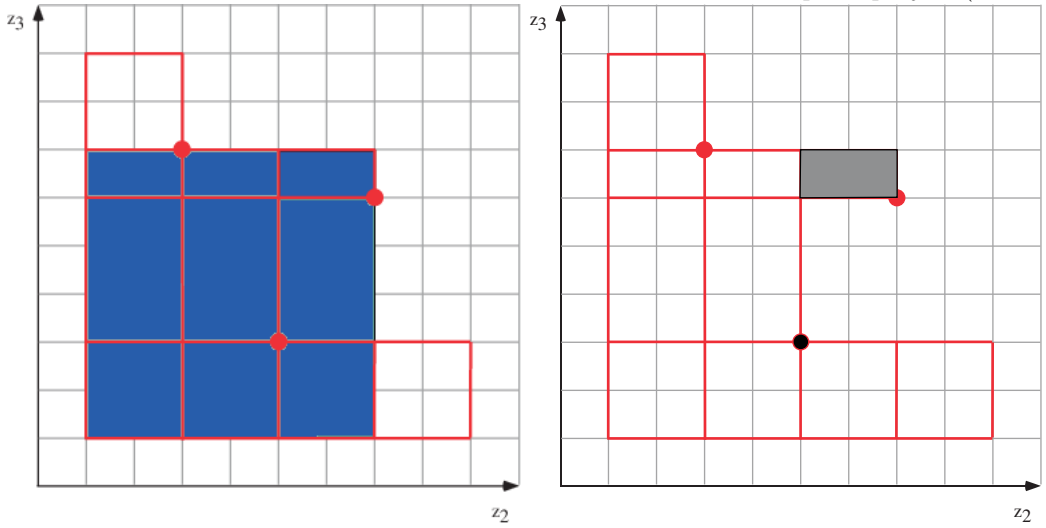
Le plus grand rectangle (bleu) est choisi pour la résolution suivante. Le choix du rectangle étudié se fait à partir du rectangle avec la plus grande aire.
Les contraintes $z_3 \leq 9 - 1$ et $z_2 \leq 7 - 1$ sont ajoutées, et on trouve le nouveau point projeté :



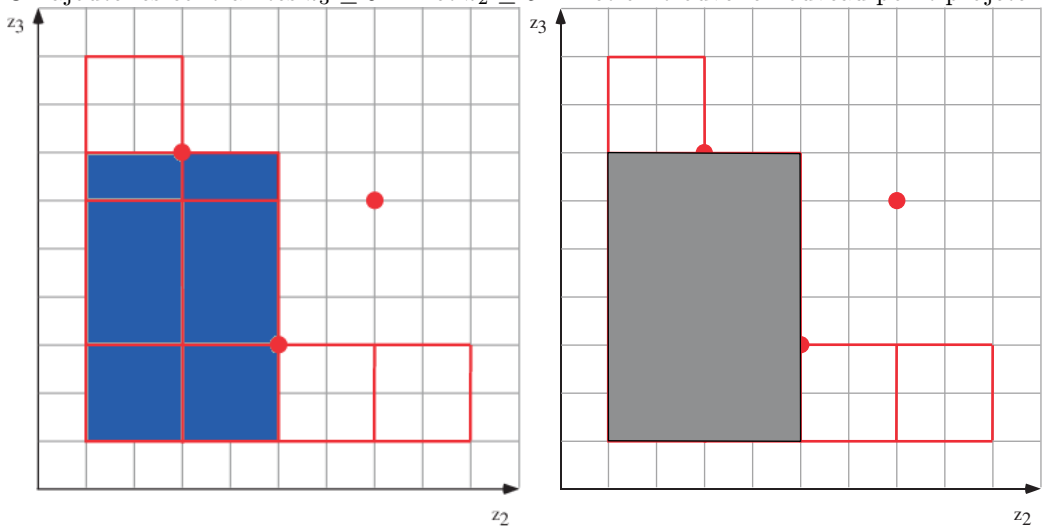
On choisit ensuite le plus grand rectangle(bleu) pour la résolution suivante.
On ajoute les contraintes $z_3 \leq 6 - 1$ et $z_2 \leq 9 - 1$ et on trouve le nouveau point projeté :



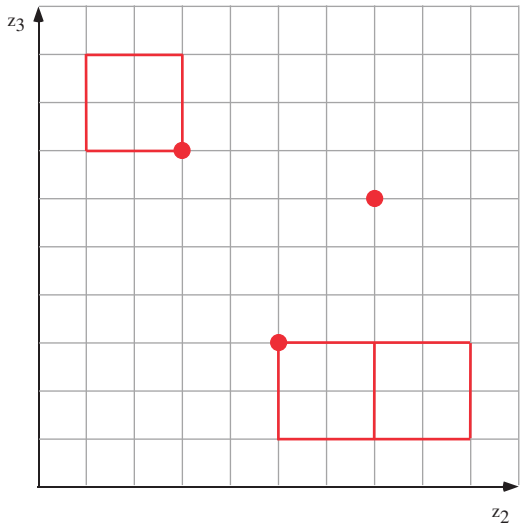
On choisit ensuite le plus grand rectangle(bleu) pour la résolution suivante.
On ajoute les contraintes $z_3 \leq 7 - 1$ et $z_2 \leq 7 - 1$ et on trouve le nouveau point projeté (redondant) :



Le plus grand rectangle ets encore choisi (bleu) pour la résolution suivante.
On ajoute les contraintes $z_3 \leq 9 - 1$ et $z_2 \leq 5 - 1$ et on trouve le nouveau point projeté :



La résolution se termine lorsqu'il n'y a plus de rectangles à explorer.



4.1 Algorithmes

Dans les algorithmes suivants, une liste L de rectangles r est utilisée. La variable r correspond au point supérieur droit du rectangle.

Algorithme 1 : Kirlik et Sayin

L $\leftarrow R(y^I, y^{AI})$ `// Ajout du rectangle de départ dans la liste`
model $\leftarrow$ monoobjective model for z1
tant que $L \neq \varnothing$ **faire**
  $r \leftarrow$ getrect(L)
  delete(L,r)
  fix(model, $z_2 \leq r_2 - 1$)`// ajout des contraintes sur les objectifs`
  fix(model, $z_3 \leq r_3 - 1$)
  $x \leftarrow$ optimize(model)
  Si *Il existe une solution optimale* x **alors**
    fix(modeldouble, $z_1 \leq z_1(x)$)`// contraintes sur les objectifs du deuxième modèle`
    fix(modeldouble, $z_2 \leq r_2 - 1$)
    fix(modeldouble, $z_3 \leq r_3 - 1$)
    $x^* \leftarrow$ optimize(modeldouble)
    makerect(L,r,$z(x^*)$)`// création des nouveaux rectangles`
    push(sols,x^*)
  fin
fin
retourner sols

Algorithme 2 : getrect(L,Y^I)

$r \leftarrow$ Plus grand rectangle de L `// findmax(map(`$x \rightarrow (Y_1^I - x_1)^2 + (x_2 - Y_2^I)^2$`,L))`
retourner r

Algorithme 3 : makerect(L,r,Y)

$isnewr1 \leftarrow true$
$isnewr2 \leftarrow true$
pour $j \in 1 : size(L)$ **faire**
    // test de redondance des nouveaux rectangles
    $isnewr1 \leftarrow isnewr1 \wedge \neg(L[j]_1 = Y_2 \wedge L[j]_2 > r_2)$
    $isnewr2 \leftarrow isnewr2 \wedge \neg(L[j]_2 = Y_1 \wedge L[j]_1 > r_1)$
fin
Si $isnewr1$ **alors**
    push(L,(Y_2,r_2))
fin
Si $isnewr2$ **alors**
    push(L,(r_1,Y_1))
fin

5 Résultats Mono-objectifs

Nous avons dans un premier temps essayé une scalarisation, en prenant des poids de l'ordre $(10^6, 10^3, 1)$. Cependant, il nous a été impossible d'obtenir des solutions avec cette démarche.

Malheureusement, l'approche exacte ne nous permet pas de résoudre l'intégralité des instances. Voici les instances sur lesquelles nous avons pu obtenir des résultats exacts (en mono-objectif).
Les cases vertes correspondent à une résolution optimale et ont donc conséquemment un gap de 0%. Les cases oranges correspondent à des résolutions où le time limit a été atteint (time limit fixé à 1 heure).
En complément :

— La valeur indiquée en rouge signifie que le temps limite a été atteint mais que la valeur minimale sur les peintures est minimale (cette valeur minimale est explicitée dans la section suivante).

— Les valeurs encadrées en noirs correspondent quant à elles à des résolutions optimales et dont la valeur minimale sur les peintures est aussi obtenue.

		z1:peinture			z2:RatioHP			z3:RatioLP		
INSTANCES A		zInt	gap	tps	zInt	gap	tps	zInt	gap	tps
1	022_3_4_EP_RAF_ENP	16	81%	TL	0	0%	68s	0	0%	13s
2	022_3_4_RAF_EP_ENP	10	20%	TL	0	0%	30s	0	0%	13s
9	039_38_4_EP_RAF_ch1	137	99%	TL	1170	100%	TL	0	0%	717s
13	064_38_2_EP_RAF_ENP_ch1	146	96%	TL	0	0%	1101s	724	0%	806s
14	064_38_2_EP_RAF_ENP_ch2	30	46%	TL	0	0%	12s	0	0%	5s
15	064_38_2_RAF_EP_ENP_ch1	161	96%	TL	0	0%	321s	724	0%	328s
16	064_38_2_RAF_EP_ENP_ch2	31	64%	TL	0	0%	4s	0	0%	5s

		z1:peinture			z2:RatioHP			z3:RatioLP		
INSTANCES B		zInt	gap	tps	zInt	gap	tps	zInt	gap	tps
1	022_EP_ENP_RAF_S22_J1	66	93%	TL	0	0%	73s	0	0%	49s
2	022_EP_RAF_ENP_S22_J1	29	93%	TL	0	0%	41	0	0%	82s
3	022_RAF_EP_ENP_S22_J1	14	42%	TL	0	0%	32s	0	0%	65s
16	028_ch2_EP_ENP_RAF_S23_J3	4	0%	3s	0	0%	0s	68	0%	0s
17	028_ch2_EP_RAF_ENP_S23_J3	4	0%	2s	0	0%	0s	68	0%	0s
18	028_ch2_RAF_EP_ENP_S23_J3	4	0%	12s	0	0%	0s	68	0%	0s
22	035_ch1_EP_ENP_RAF_S22_J3	6	0%	8s	61	0%	0s	49	0%	0s
23	035_ch1_EP_RAF_ENP_S22_J3	6	0%	6s	61	0%	0s	49	0%	0s
24	035_ch1_RAF_EP_ENP_S22_J3	6	0%	5s	61	0%	0s	49	0%	1s
25	035_ch2_EP_ENP_RAF_S22_J3	7	0%	16s	270	0%	2s	72	0%	1s
26	035_ch2_EP_RAF_ENP_S22_J3	7	0%	11s	270	0%	2s	72	0%	1s
27	035_ch2_RAF_EP_ENP_S22_J3	7	0%	163s	270	0%	2s	72	0%	1s
38	048_ch2_EP_RAF_ENP_S22_J3	107	98%	TL	0	0%	1380s	0	0%	346s
39	048_ch2_RAF_EP_ENP_S22_J3	106	99%	TL	0	0%	1154s	0	0%	1242s
43	064_ch2_EP_ENP_RAF_S22_J4	36	80%	TL	0	0%	12s	0	0%	5s
44	064_ch2_EP_RAF_ENP_S22_J4	36	83%	TL	0	0%	9s	0	0%	14s
45	064_ch2_RAF_EP_ENP_S22_J4	36	69%	TL	0	0%	10s	0	0%	9s

		z1:peinture			z2:RatioHP			z3:RatioLP		
INSTANCES X		zInt	gap	tps	zInt	gap	tps	zInt	gap	tps
1	022_RAF_EP_ENP_S49_J2	14	50%	TL	0	0%	67s	0	0%	123s
6	028_CH2_EP_ENP_RAF_S51_J1	3	0%	0s	0	0%	0s	0	0%	0s
7	029_EP_RAF_ENP_S49_J5	57	98	TL	0	0%	112s	0	0%	131s
9	034_VU_EP_RAF_ENP_S51_J1_J2_J3	10	40%	TL	1	0%	10s	12	0%	2s
10	035_CH1_RAF_EP_S50_J4	5	0%	0s	9	0%	0s	0	0%	0s
11	035_CH2_RAF_EP_S50_J4	6	0%	177s	0	0%	0s	0	0%	0s
17	064_CH2_EP_RAF_ENP_S49_J4	26	53%	TL	0	0%	6s	0	0%	3s
18	655_CH1_EP_RAF_ENP_S51_J2_J3_J4	31	32%	TL	0	0%	0s	0	0%	0s
19	655_CH2_EP_RAF_ENP_S52_J1_J2_S01_J1	28	32%	TL	138	0%	0s	0	0%	0s

Au regard de nos résolutions, il en résulte qu'avec l'approche exacte, ce sont les peintures qui posent le plus de problème à résoudre à l'optimalité. Ainsi, dans l'ε-contrainte, c'est cet objectif qui sera passé en contrainte et les composants de hautes/basses priorités sur les composants en objectif.

	Approche Lex basique	**Approche Lex allégée**	
Objectifs	z_{CHP}, z_{CLP}, z_{Paint}	z_{CHP}, z_{CLP}	z_{Paint}
Set A	0	6	7
Set B	9	17	16
Set X	3	10	8

TABLE 1 – Nombres d'instances résolues par set et approche

6 Améliorations

Suite aux résultats précédents, nous avons pu voir qu'une unique résolution est très consommatrice en temps. Afin de pouvoir réaliser une méthode multi-objectif basée sur des résolutions de notre problème mono-objectif, nous avons cherché à utiliser les particularités du solveur mono-objectif Gurobi et de notre problème.
En effet, lors d'une résolution de notre problème mono-objectif, nous avons pu remarquer que la qualité de la solution initiale était très mauvaise par rapport à l'optimum que nous trouvions. Notons aussi que la relaxation

du problème donne aussi une relaxation de très mauvaise qualité puisqu'elle est, elle aussi, éloignée de l'optimum. Afin d'avoir les solutions dans un temps raisonnable, l'objectif est donc de mettre en place des stratégies d'amélioration du temps de résolution en améliorant la borne inférieure et la borne supérieure connues par le solveur. Ce dernier se base sur un branch and bound et par conséquent, améliorer ces bornes permet de réduire le nombre de noeuds à explorer. Nous présentons, ici, les méthodes de réduction des bornes que nous avons pu mettre en place ainsi que d'autres perspectives d'améliorations.

6.1 Borne inférieure : Le nombre minimum de changements de peintures

Parmi les résultats obtenus précédemment, nous avons pu remarquer que le problème mono-objectif avec comme fonction objectif le nombre de changements de peintures est le plus long à résoudre. Pourtant, d'après les contraintes du modèle, nous pouvons remarquer que trois données ont un impact sur le changement de couleur : le nombre de peintures différentes, le carnet de commandes avec le nombre de voitures pour chaque couleur ainsi que l'indication des couleurs pour le jour précédent.
Nous proposons donc une manière d'obtenir une borne inférieure plus contraignante que celle de Gurobi pour le problème mono-objectif avec uniquement la somme des violations de peintures.

Dans le cadre du challenge, l'indication des couleurs pour le jour précédent n'est pas prise en compte. Nous allons donc déjà proposé une première façon de calculer le nombre minimum de changements de couleur sans cette indication.

Dans un premier temps, notons qu'il y a nécessairement autant de changements de couleurs que de nombre de couleurs différentes. Cependant, cette borne reste très insuffisante pour beaucoup de problèmes. Il est possible d'avoir peu de couleurs pour beaucoup de voitures. Nous allons donc prendre en compte le nombre maximal de rafale de peinture s. Nous construisons donc des blocs de taille maximale s de voitures de même couleur de peinture de sorte à remplir ces blocs au maximum, sachant que le dernier bloc correspondra à un reste. Cette construction correspond donc à réaliser une division euclidienne du nombre de voiture de couleur i par la taille limite s d'un bloc, dont le quotient q sera le nombre de blocs complet et, le reste r nous permet de déterminer s'il existe un bloc incomplet en plus. Nous aurons donc le nombre de bloc de la couleur i. Ainsi, nous pouvons déterminer ce nombre pour chaque couleur.

Comme un bloc d'une couleur sera systématiquement suivit par un bloc d'une autre couleur, nous allons donc mettre bout à bout des blocs, toujours en mettant côte à côte des blocs de couleurs différentes. En créant cette séquence, nous remarquons donc que nous avons deux possibilités, soit le nombre de voitures d'une certaine couleur est trop important et une telle séquence est irréalisable, soit il est possible d'en créer une.

Comme nous prenons en compte la taille limite d'un bloc de couleur comme une contrainte dure, il est possible de se rendre compte si le problème est infaisable. En effet, si la couleur la plus fréquente a un nombre de blocs supérieur de plus de 1 à la somme des blocs de toutes les autres couleurs, alors deux blocs de même couleurs vont forcément se retrouver côte à côte. Nous avons là un problème infaisable.

Dans le cas inverse, il est possible d'envisager une solution réalisable, et comme nous avons réussi à agencer les blocs selon la contrainte de couleur, alors le nombre de changement de couleur correspond au nombre de passage d'un bloc à l'autre, soit du nombre de blocs total moins 1.

Algorithme 4 : Borne du nombre minimum de changements de couleur

Entrées : d le nombre de voiture pour chaque couleur, s la taille limite d'un bloc
Sorties : S le nombre de changements minimum de couleurs

```
1  ;
   // Calcul du nombre de bloc pour chaque couleur
2  pour i ∈ [1, ..., taille(d)] faire
       // Quotient et reste de la division euclidienne
3      q, r ⟵ d_i ÷ s;
       // Ajout dans la somme
4      S ⟵ S + q ;
5      Si r > 0 alors
6          S ⟵ S + 1 ;
7      fin
8  fin
9  S ⟵ S - 1 ;
10 Retour S
```

Nous avons donc une borne qui est de meilleure qualité. Dans le cadre des expérimentations, ce calcul revient à trouver la valeur lexicographique. Comme nous sommes sous l'hypothèse que les couleurs des voitures du jour précédent sont prises en compte, ce nombre n'est qu'une borne inférieure, cependant sans cette hypothèse, cette valeur est optimale.

Avec notre hypothèse, il serait possible de vouloir encore améliorer cette borne. Cependant, afin de calculer la meilleure séquence pour réduire le nombre de blocs, il faut maintenant prendre au cas par cas les données du jour précédent avec leurs blocs et répartitions de couleur. Le problème devient tout de suite plus ardu puisqu'il est possible que les blocs de couleurs ne soient pas ordonnées comme la séquence présentée ci-dessus. Comme les expérimentations montraient que nous trouvions très fréquemment la valeur optimale sans prendre en compte le jour précédent, nous avons gardé cette borne. Cela permet au solveur de s'arrêter plus vite, puisqu'il permet de réduire le nombre de noeuds à explorer.

Ce nombre est aussi très intéressant dans le cadre de l'ε-contrainte. En effet, les expérimentations ont montré qu'il est nécessaire de passer la fonction objectif sur les changements de peintures en contrainte. De ce fait, connaître le nombre minimum de changements de peintures permet de rajouter une contrainte d'arrêt pour l'ε-contrainte dédiée à notre problème. Si la dernière solution trouvée implique avoir un nombre égale à la borne, il est alors inutile de poursuivre l'ε-contrainte. En effet, cela permet de supprimer la dernière étape où le solveur vérifie qu'il n'existe plus aucune solution réalisable. Cette dernière résolution est très coûteuse et pour pouvoir obtenir des résultats sur un plus grand nombre d'instances, il a été utile de rajouter cette condition d'arrêt.

6.2 Borne supérieure : une solution admissible de qualité

Nous avons noté que les heuristiques de Gurobi ne permettaient pas de trouver des solutions proches de l'optimal dans le cadre de notre problème. C'est une piste d'amélioration, d'autant plus pour les méthodes multi-objectifs qui réalisent plusieurs appels au solveur avec le problème mono-objectif.

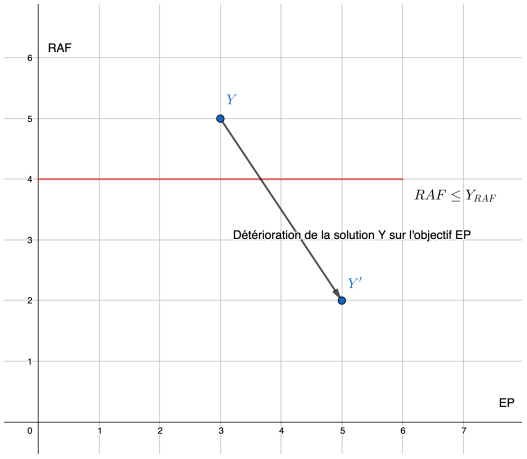
En minimisation, n'importe quelle solution admissible fait office de borne supérieure. À chaque itération de l'ε-contrainte, un ϵ_k est fixé pour borner la fonction objectif passée en contrainte. On résout alors le programme linéaire correspondant. Le fait est qu'à chacune de ces itérations, le solveur réinitialise de nouvelles bornes primales et duales dans son Branch and bound sans exploiter les solutions obtenues lors des précédentes itérations.

Afin d'améliorer la résolution, il serait possible de fournir à Gurobi une solution admissible dont la valeur ferait office de borne supérieure pour chaque itération de l'ϵ contrainte.

Dans l'ε-contrainte, la solution optimale de l'itération précédente n'est pas réalisable pour le problème de l'itération actuelle. Pour pallier ce problème, l'idée est d'utiliser une heuristique de dégradation qui part de la solution précédente. En dégradant la valeur de la fonction objectif sur les peintures, ou bien en améliorant la valeur de la fonction objectif sur les composants, il est possible de proposer une solution partielle au solveur

pour qu'il la reconstruise et l'utilise. Il est aussi possible de créer une heuristique complète reconstruisant elle-même la solution et la fournir au solveur. Créer une heuristique complète permettrait de réaliser une méthode spécifique et efficace pour notre problème, contrairement à l'heuristique de construction du solveur sur lequel aucune information n'est donnée.

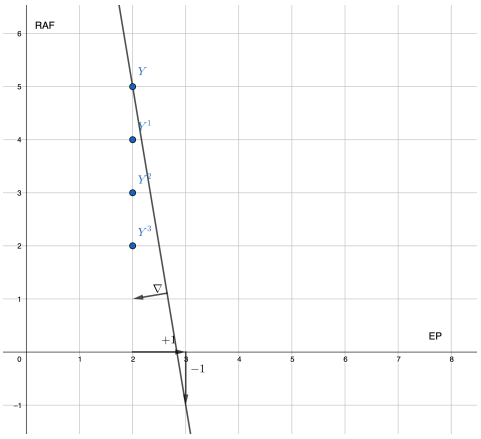
Remarque : Le recours aux solutions obtenues par nos camarades travaillant sur des approches heuristiques pour la résolution du CSP a plusieurs fois été envisagé et implémenté. Cependant, une hypothèse sur la journée la purge réalisée sur les peinture entre les productions de la veille et du jour considéré ne nous permettent pas d'intégrer leurs solutions comme borne ; elles sont potentiellement non admissibles pour notre modèle.



6.3 Les solutions faiblement efficaces en bi-objectif

En général dans les approches multi-objectif, nous n'avons pas d'intérêts à conserver les solutions faiblement efficaces. En plus d'être d'utilité moindre : l'énumération des points faiblement non dominés peut rajouter un temps d'exécution non négligeable.

Faire une résolution mono-objectif sur les composants permet de récupérer une solution lexicographiquement optimale sur un objectif mais potentiellement faiblement efficace pour notre problème bi-objectif. Nous voulons retrouver une solution efficace. Pour ce faire, il est possible de réaliser, par scalarisation, une autre résolution de problème mono-objectif en choisissant judicieusement les poids. De cette solution de coordonnées $Y = (y_1, y_2)$, un autre point non réalisable de coordonnées $P = (y_1 + 1, -1)$ est construit. L'orthogonale à la droite construite passant par ces deux points nous donne un vecteur Δ (cf figure ci dessous).
Ce vecteur nous permet de déduire une pondération afin de trouver la solution lexicographiquement optimale et efficace. Par la suite, il est possible de commencer toutes nos résolutions à partir de points non dominées sans énumérer les points faiblement non dominées.



7 Résultats Multi-objectifs

Nous avons commencé les tests par les plus petites instances que nous avions réussi à résoudre de manière lexicographique. Nous avons été très surpris, lors des premières résolutions, de trouver seulement le point idéal dans l'ensemble des points non dominés.

Après avoir réalisé ce constat, nous avons mis en place une procédure de test d'admissibilité pour déterminer si cette observation se reproduisait sur d'autres instances. Nous avons trouvé deux instances parmi les plus petites où le point idéal était non admissible. Nous avons alors lancé notre méthode tri-objectif sur ces instances pour trouver les ensembles des points non dominés suivants :

```
   julia> rune3o(2,16)
Data/Instances_Set_B/028_ch2_EP_ENP_RAF_S23_J3/
Cars(J-1) = 27
Cars(J) = 65
Cpmt = 7
Color = 3
Première solution : (0,194,386)
Solution double (EP,RAF,ENP) : (0,5,69)
Première solution : (0,194,68)
Solution double (EP,RAF,ENP) : (0,6,68)
Première solution : (0,4,386)
Solution double (EP,RAF,ENP) : (0,4,71)
La résolution se termine avec 3 solutions, en enlevant les faiblement efficaces
```

La première résolution nous retourne un point (0,194,386) qui est potentiellement faiblement efficace. En effet, lors de la deuxième résolution, une solution avec la même valeur sur le premier objectif et de meilleure valeur sur les deux autres est trouvée : (0,5,69). Si nous n'avions pas implémenté la double résolution, la méthode aurait coupé sur 386 et on aurait très certainement obtenu la solution (0,194,385) lors de l'itération suivante.

On observe un mécanisme similaire avec les points : (0,194,68) et (0,6,68) ce qui nous donne bien les trois points non dominés suivants (0,5,69), (0,194,68) et (0,6,68).

Nous notons que le premier objectif (composants de haute priorité) est toujours 0. En effet, cet objectif a été choisi comme objectif à optimiser car c'est le moins contraignant. En pratique, nous retrouvons souvent la valeur lexicographique du premier objectif dans les solutions de l'ensemble des points non dominés.

```
   julia> rune3o(2,22)
Data/Instances_Set_B/035_ch1_EP_ENP_RAF_S22_J3/
Cars(J-1) = 5    Cars(J) = 128    Cpmt = 4    Color = 7
Première solution : (61,895,256)
Solution double (EP,RAF,ENP) : (61,43,61)
Première solution : (61,895,60)
Solution double (EP,RAF,ENP) : (61,44,60)
Première solution : (61,895,59)
Solution double (EP,RAF,ENP) : (61,47,59)
Première solution : (61,895,58)
Solution double (EP,RAF,ENP) : (61,47,58)
Première solution : (61,895,57)
Solution double (EP,RAF,ENP) : (61,50,57)
Première solution : (61,895,56)
Solution double (EP,RAF,ENP) : (61,50,56)
Première solution : (61,895,55)
ERROR: Gurobi.GurobiError(10005, "Unable to retrieve attribute 'X'")
```

Ce deuxième exemple présente des résultats similaires au premier à ceci près que les résolutions sont beaucoup plus lourdes. En effet, nous approchons de la limite de taille d'instances qui peuvent être résolues à l'exact. Nous faisons l'hypothèse que la complexité du problème introduit des erreurs d'imprécisions numériques lors de la résolution de Gurobi, ce qui expliquerait les solutions faiblement dominées et l'échec de la preuve d'infaisabilité à la fin de la résolution.

8 Conclusion

L'opportunité de travailler sur de réelles instances nous a permis de comprendre les problématiques rencontrées dans le milieu industriel par rapport aux problèmes académiques que nous avons l'habitude d'étudier dans notre cursus. En effet, les données réelles renferment bien plus de subtilités que l'on pourrait exploiter afin d'améliorer les performances de nos modèles et méthodes.

L'approche exacte, bien qu'en difficulté sur les grandes instances, nous a permis de mettre en lumière l'objectif problématique, à savoir les peintures, qui rendait le problème complexe à résoudre et de détecter l'objectif le moins contraignant, les hautes priorités. La résolution sur de petites instances nous montre qu'il existe souvent moins d'une dizaine de solutions efficaces lorsque l'ensemble des points n'est pas dominé par une unique solution.

Avec le recul et les résultats que nous avons aujourd'hui, nous pouvons émettre l'hypothèse que le développement d'un modèle dans un paradigme de résolution différent (modélisation SAT, programmation par contrainte, ...) pourrait être une piste à explorer. Une analyse des données en amont de production nous permettrait d'obtenir certaines indications, comme par exemple le nombre minimum de changement de couleurs sur un film, afin de déterminer un paradigme potentiellement plus efficace. En effet, des heuristiques spécifiques pourraient être utilisées en parallèle de la résolution exacte du modèle comme par exemple des heuristiques énumératives ou autres méthodes développées par nos camarades, sous couvert de considérer le même ensemble d'hypothèses au départ.

Références

[1] Gokhan Kirlik et Serpil Sayin. "A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems". In : *European Journal of Operational Research* 232 (fév. 2014), p. 479-488. DOI : 10.1016/j.ejor.2013.08.001.

[2] Matthias Prandtstetter et Günther R. Raidl. "An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem". In : *European Journal of Operational Research* 191.3 (2008), p. 1004-1022. ISSN : 0377-2217. DOI : https://doi.org/10.1016/j.ejor.2007.04.044. URL : http://www.sciencedirect.com/science/article/pii/S0377221707004535.

[3] Christine Solnon et al. "The car sequencing problem : Overview of state-of-the-art methods and industrial case-study of the ROADEFâ2005 challenge problem". In : *European Journal of Operational Research* 191.3 (2008), p. 912-927. ISSN : 0377-2217. DOI : https://doi.org/10.1016/j.ejor.2007.04.033. URL : http://www.sciencedirect.com/science/article/pii/S0377221707004468.