Université Libre de Bruxelles
Faculté des Sciences
Département d'informatique

Rapport

Luyckx Marco 496283

Marques Correia Tiago 495305

Professeur: Renaud Chicoisne

Cours : Algorithmique et recherche opérationnelle INFO-F310

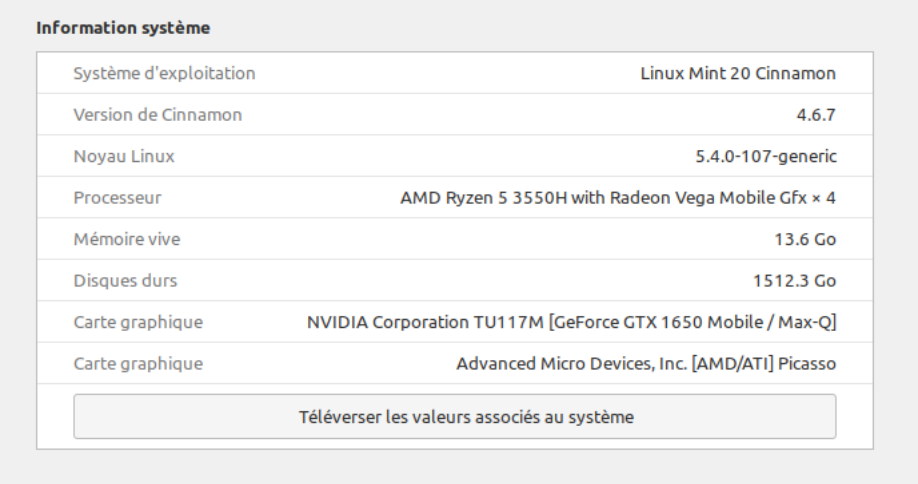
Remis en avril 2022

Année académique 2021-2022

Table des matières

1 Paramètres

Tous nos tests ont été effectués sous Linux Mint 20 Cinnamon.
Nous avons automatisé nos tests grâce à un autre programme : automat.py.
Tous les résultats de toutes les instances données pour chaque p sont dans un fichier nommé resultats.txt. Il a servi comme pilier dans nos conclusions. Nous avons "pseudo-vérifié" nos résultats pour $p = 0$ avec une heuristique qui sera détaillée plus tard. Notre pseudo-vérification se base sur un constat simple : si le nombre de boîtes maximum trouvé par l'heuristique est plus petit que celui trouvé par notre programme linéaire, alors il doit y avoir une erreur. Le nom du programme est heuristique.py.
Voici tous les paramètres de notre configuration qui pourraient vous intéresser :



2 Modélisation

2.1 Formulation

Ensemble :

T = types de produits

B = boîtes

I = incompatibilités

Paramètres :

s_t = taille du type du produit t

n_t = nombre du type du produit t

q = capacité des boîtes b

M = un nombre très grand - constante artificielle big-M

mo_t = nombre maximum de boîtes dans lequel un type de produit t peut être présent

Variables de décision :

$x_{b,t} \in \mathbb{Z}^+$, nombre de produits de type $t \in T$ mis dans la boîte $b \in B$

$y_b \in \{0,1\}$, telles que

$$y_b = \begin{cases} 1 & \text{si la boîte b} \in B \text{ est utilisée} \\ 0 & \text{sinon} \end{cases}$$

$z_{b,t} \in \{0,1\}$, telles que

$$z_{b,t} = \begin{cases} 1 & \text{si la boîte b} \in B \text{ contient un produit de type t} \in T \\ 0 & \text{sinon} \end{cases}$$

Formulation :

$$min \quad \sum_{b \in B} y_b \tag{1}$$

$$s.t \quad \sum_{b \in B} x_{b,t} = n_t \qquad \forall t \in T \tag{2}$$

$$\sum_{t \in T} s_t x_{b,t} \leqslant q y_b \qquad \forall b \in B \tag{3}$$

$$x_{b,t} \leqslant M z_{b,t} \qquad \forall b \in B, t \in T \tag{4}$$

$$z_{b,t} \leqslant x_{b,t} \qquad \forall b \in B, t \in T \tag{5}$$

$$\sum_{b \in B} z_{b,t} \leqslant m o_t \qquad \forall t \in T \tag{6}$$

$$\sum_{e \in i} z_{b,e} \leqslant 1 \qquad \forall i \in I, b \in B \tag{7}$$

$$x_{b,t} \in \mathbb{Z}^+ \qquad \forall b \in B, t \in T \tag{8}$$

$$y_b \in \{0,1\} \qquad \forall b \in B \tag{9}$$

$$z_{b,t} \in \{0,1\} \qquad \forall b \in B, t \in T \tag{10}$$

2.2 Explications

L'objectif **(1)** permet de fixer le but de notre problème. Dans notre cas, nous cherchons à minimiser le nombre de boîtes utilisées. Nous modélisons cela par la somme d'une variable binaire sur l'utilisation d'une boîte $b \in B$.
La contrainte **(2)** permet de s'assurer que chaque produit sera présent dans une boîte. Nous modélisons cela avec la somme du nombre de produit de type $t \in T$ dans l'ensemble des boîtes $b \in B$ doit être exactement égal au nombre total de type de produit $t \in T$.
La contrainte **(3)** permet de s'assurer que la taille des produits mis dans une boîte ne dépasse pas la capacité maximale de la boîte. Nous modélisons ça par une somme de la taille du produit $t \in T$ multiplié par leur nombre dans la boîte $b \in B$ et ce, pour tout les produits $t \in T$ dans chacune des boîtes $b \in B$. Si la boîte est utilisée, on s'assure que cette somme est plus petite ou égale à la capacité des boîtes. Si elle est vide, alors il ne peut y avoir aucun produit dedans.
Les contraintes **(4)** et **(5)** sont complémentaires.
La contrainte **(4)** permet de forcer $x_{b,t} = 0$ dès que $z_{b,t} = 0$. Dans notre problème, on modélise une contrainte big-M. Celle-ci doit être choisie pour être sans effet lorsque $z_{b,t} = 1$. Nous l'avons donc représentée par la quantité de produits la plus grande parmi les $t \in T$ types de produits possibles.
Il y a deux possibilités :

1. Si la boîte $b \in B$ contient au moins un produit $t \in T$ (donc que $z_{b,t} = 1$), alors le nombre de produit $(x_{b,t})$ doit au **maximum** être égal au nombre total de produits de type $t \in T$;
2. Si la boîte $b \in B$ ne contient pas de produit $t \in T$ (donc que $z_{b,t} = 0$), alors il faut s'assurer que le nombre de produit $(x_{b,t})$ soit égal à 0.

Le seul souci qu'il y a encore avec cette contrainte est que, même si $z_{b,t} = 1$, rien ne force la variable $x_{b,t}$ à être différente de 0, ce qui représenterait une incohérence puisque cela signifierait que : "La boîte $b \in B$ contient un produit $t \in T$ **MAIS** la variable $x_{b,t}$ qui contient le nombre de produits $t \in T$ mis dans cette boîte $b \in B$ est égal à 0". Il nous faut donc une contrainte supplémentaire qui est la contrainte **(5)**. Lorsque $z_{b,t} = 1$, nous nous assurons que la variable $x_{b,t}$ correspondante soit **au moins égale** à 1. Nous venons donc de résoudre le souci énoncé juste avant. De plus, on remarque bien évidemment que cette contrainte n'a aucune influence sur les autres cas de la contrainte **(4)**.
La contrainte **(6)** modélise la contrainte du $p = 1$ qui permet de mettre certains produits $t \in T$ dans au plus mo_t boîtes. Nous la modélisons par une somme de variables binaires pour chaque type de produit $t \in T$ dans chacune des boîtes $b \in B$. Cette somme doit être inférieure au nombre maximum de boîtes dans lequel un type de produit $t \in T$ peut être présent. Si nous devions l'exprimer en français: "Nous allons regarder, pour chaque type de produit, s'il est présent dans un nombre de boîtes inférieur ou égal au maximum autorisé pour ce type."
La contrainte **(7)** modélise la contrainte du $p = 2$ qui interdit à des produits incompatibles d'être dans la même boîte. Nous modélisons cela par une somme de variables binaires pour chaque incompatibilité $i \in I$ et ceci dans chacune des boîtes. Cette somme doit être inférieure

ou égale à 1 pour qu'il n'y ait pas deux produits incompatibles dans la même boîte. Si nous devions l'énoncer en français: "Pour chaque incompatibilité $i \in I$ et pour chaque boîte $b \in B$, nous vérifions si l'incompatibilité i n'est pas présente dans la boîte b."
Les contraintes **(8)**, **(9)** et **(10)** permettent de fixer les valeurs possibles que pourront prendre nos variables de décision.

Pour aider le lecteur à situer ces différentes contraintes dans le code, nous avons indiqué en commentaire à quelle contrainte se rapportait chaque bout de code.

3 Analyse des résultats

L'analyse est divisée en 3 parties, une pour chaque p. Nous expliciterons, dans chacun des cas, le pourcentage de réponses optimales, les moyennes de temps de résolution avec et sans les time-out, les temps de résolutions moyens par nombre total de produits ainsi que quelques autres remarques propres au p.
Tous nos résultats sont consultables dans un fichier texte : resultats.txt.

3.1 Sans contrainte : p = 0

Pour choisir un nombre de boîtes, nous supposons, qu'au maximum, chaque produit ira dans une boîte différente. Nous pouvons donc prendre le nombre total des produits comme nombre de boîtes maximum. Une optimisation possible est de prendre le nombre de boîtes calculé par une fonction heuristique qui nous donnerait une valeur plafond. Nous avions d'abord décidé d'utiliser une librairie toute faite nommée binpacking. Cependant, après un échange de mail avec Monsieur De Boeck Jérôme, nous avons conclu qu'il était préférable d'implémenter notre propre fonction heuristique pour ne pas utiliser une "boîte noire" dans laquelle nous ne savons pas ce qu'il se passe. En nous basant sur le pseudo-code [1], nous avons implémenté cette fonction heuristique FirstFit (dans notre code, elle s'appelle "greedy()") et nous avons constaté un accroissement notable de la performance (voir ci-dessous).

Pourcentage de réponses optimales :
Premièrement, sans l'optimisation de l'heuristique, sur les 105 instances que comporte le dossier, nous avions 63% de réponses optimales trouvées. Avec l'optimisation, ce pourcentage grimpe à 87%, ce qui est un amélioration significative.

Moyenne des temps de résolution :
En ce qui concerne les temps d'exécution, la moyenne est de 95 secondes par fichier pour les 105 fichiers testés, en prenant en compte les fichiers qui arrivent à 10 minutes. Si nous ne prenons pas en compte les 16 fichiers qui sont arrivés à la limite de temps (et qui donc donnent une réponse non-optimale), la moyenne passe à 18 secondes.

Temps de résolution moyen par nombre total de produits :

Moyenne pour p = 0 par nombre total de produits	
Nombre total de produits	Temps de résolution moyen
10	0 secs
12	0 secs
14	47 secs
16	0 secs
18	0 secs
20	82 secs
22	82 secs
24	457 secs
26	200 secs
28	0 secs
30	0 secs
32	400 secs
34	400 secs
36	0 secs
38	537 secs
40	120 secs

On remarque une augmentation du temps moyen quand le nombre de produits augmente et c'est logique : plus il y a de produits, plus il y aura de variables et donc de calculs à effectuer pour trouver la solution optimale. En analysant les résultats plus en profondeur, on s'aperçoit que, pour les instances qui ont des temps moyens élevés, la réponse de la fonction heuristique était déjà très grande.
Prenons par exemple l'instance 38 qui est une des plus grandes instances. On remarque que les réponses données par id respectifs pour la fonction heuristique sont : 37, 29 et 18 ce qui confirme notre remarque précédente.
Par ailleurs, la taille des produits dans les instances influe énormément sur la vitesse de résolution. Quand les tailles des produits "s'emboîtent" correctement entre elles, c'est-à-dire qu'elles remplissent au maximum, ou quasiment au maximum, la capacité disponible; on constate que le solveur trouve une réponse plus rapidement dans la majorité des cas.

Pour mieux comprendre ce que nous essayons de vous exposer, voici un exemple sur l'instance 30_4_2_3 :
1) Il y a 16 produits de taille 7
2) Il y a 1 produit de taille 9
3) Il y a 1 produit de taille 27
4) Il y a 12 produits de taille 16
Après un petit coup d'œil, on remarque directement une combinaison parfaite : $16+16+16+16+27+9$ qui nous donne exactement 100. Et ensuite, il suffit simplement de maximiser l'espace dans les autres boîtes avec les produits restants. On voit donc directement qu'il existe une multitude de solutions possibles à partir de celle-là (permutations de 16 par exemple).

Pour certains fichiers, notre programme linéaire a trouvé une réponse optimale plus petite que celle de notre programme heuristique : heuristique.py.

Voici un exemple pour le fichier 14_8_4_1:
Réponse par heuristique : 6
Réponse par programmation linéaire : 5
Vérification :
Bin1 → 60 + 27 + 13 = 100
Bin2 → 59 + 27 + 14 = 100
Bin3 → 60 + 33 = 93
Bin4 → 60 + 15 + 15 = 90
Bin5 → 32 + 32 + 32 = 96

Un autre exemple pour le fichier 14_8_9_1:
Réponse par heuristique : 6
Réponse par programmation linéaire : 5
Vérification :
Bin1 → 58 + 21 + 21 = 100
Bin2 → 53 + 22 + 21 = 96
Bin3 → 44 + 44 = 88
Bin4 → 44 + 44 = 88
Bin5 → 21 + 21 + 21 + 30 = 93

3.2 Contrainte du nombre maximum de produits : p = 1

Pourcentage de réponses optimales :
Pour toutes les instances, nous avons 59% des réponses optimales. On constate une baisse à cause des contraintes supplémentaires qui se sont ajoutées.
Remarque : Sur les 105 fichiers testés, nous en avons obtenu 2 pour lesquels la réponse était infaisable (SOLUTION IS INFEASIBLE). Nous nous retrouvons donc au final avec 103 fichiers testés et non 105.

1) Contrainte impossible à faire pour l'instance 20_5_2_2 : 26 4 1. Si on traduit en français, cela donne : "Essayez de mettre 4 produits de taille 26 dans une seule et même boîte de capacité 100". C'est tout simplement infaisable.
2) Contrainte impossible à faire pour l'instance 40_4_2_3 : 36 17 8. Même constat ici, on ne peut pas mettre 17 produits de taille 36 dans 8 boîtes.

Moyenne des temps de résolution :
Pour les 103 fichiers, la moyenne de temps d'exécution par fichier est de 253 secondes. Si nous ne tenons pas compte des fichiers qui excèdent le temps limite, la moyenne descend à seulement 14 secondes.

Temps de résolution moyen par nombre total de produits :

Moyenne pour p = 1 par nombre total de produits	
Nombre total de produits	Temps de résolution moyen
10	0 secs
12	8 secs
14	122 secs
16	400 secs
18	600 secs
20	207 secs
22	600 secs
24	600 secs
26	600 secs
28	200 secs
30	200 secs
32	400 secs
34	600 secs
36	250 secs
38	600 secs
40	300 secs

Comme expliqué pour le $p = 0$, ce n'est pas uniquement du nombre total de produits que dépendent les temps de résolutions mais aussi de la taille des produits. Si en plus de cela, on ajoute une contrainte sur les boîtes, alors le temps de résolution s'accroît rapidement.

3.3 Contrainte du nombre maximum de produits et contrainte d'incompatibilité : p = 2

Pourcentage de réponses optimales :
Nous avons 38% de réponses optimales. De nouveau, on constate une baisse due à l'accumulation des contraintes et donc, logiquement, avec le nombre de variables qui augmente, cela provoque également une augmentation du temps de calcul.
Comme pour $p = 1$, les fichiers qui étaient déjà infaisables le sont toujours.

Moyenne des temps de résolution :
Si nous prenons tous les fichiers en compte, la moyenne de temps d'exécution par fichier s'élève à 393 secondes. Par contre, si nous ne prenons pas en compte les fichiers arrivant à la limite de temps, la moyenne est de 62 secondes par fichier.

Temps de résolution moyen par nombre total de produits :

Moyenne pour p = 2 par nombre total de produits	
Nombre total de produits	Temps de résolution moyen
10	0 secs
12	0 secs
14	411 secs
16	413 secs
18	600 secs
20	379 secs
22	410 secs
24	600 secs
26	600 secs
28	200 secs
30	200 secs
32	600 secs
34	600 secs
36	400 secs
38	600 secs
40	600 secs

Comme on pouvait s'y attendre, les temps moyens sont souvent des time-out. L'accumulation du nombre total de produits qui augmente, de la taille des produits et de l'ajout de la dernière contrainte, en plus de celle de $p = 1$, crée un gros paquet de variables supplémentaires, ce qui augmente le temps de calcul.

La seule instance qui nous "pose problème" est 38_4_2_1 car nous obtenons un "INTEGER UNDEFINED" pour celui-ci. Parmi tout le jeu d'instances données, c'est celle qui nécessiterait le plus de boîtes d'après notre fonction heuristique (37 pour $p = 0$). Une de nos hypothèses est que le solveur n'a pas encore eu assez de temps pour trouver une réponse, même non-optimale.

4 Conclusion

En conclusion, nous observons que les facteurs impactants sur les performances sont :

1. Le nombre total de produits : on constate une augmentation du temps avec le nombre de produits qui croît. Plus de produits signifie plus de variables à traiter et donc plus de temps de calcul ;
2. La taille des types de produits : nous pensons qu'une cause potentielle des time-out (et donc d'une réponse non-optimale) de certaines instances, est qu'il existe un nombre très restreint de solutions qui maximise l'utilisation de la capacité de chaque boîte de cette instance. Les tailles de produits ne s'imbriqueraient pas facilement. Autrement dit, certaines boîtes vont avoir beaucoup de "vide" (espace sans produit) car il n'existe tout simplement pas de produits qui peuvent compléter la boîte sans dépasser la capacité maximale de celle-ci ;
3. Les contraintes supplémentaires : on remarque très nettement que plus on ajoute de contraintes, plus les temps de résolution sont longs car ces contraintes ajoutent des variables et donc du temps de calcul.

C'est donc la combinaison de tous ces paramètres qui augmente le temps que met le solveur.

Pour confirmer nos dires, nous relevons que les time-out sur le $p = 0$ sont rares, que sur le $p = 1$, ils le sont déjà moins et que sur le $p = 2$, ils sont courants.

Bien évidemment, un autre facteur qui nous semblait implicite (mais qui est fondamental), est la modélisation du problème. Il est sûrement possible de réduire les temps de résolution en optimisant la modélisation du problème. Par ailleurs, les composantes informatiques de la machine sur laquelle le programme tourne jouent aussi un rôle primordial.

5 Références

[1] Rieck Bastian. Basic Analysis of Bin-Packing Heuristics. Reperé le 11/04/2022 à `https://bastian.rieck.me/research/Note_BP.pdf`