UNIVERSITÉ PIERRE ET MARIE CURIE

RÉSOLUTION DE PROBLÈMES

PROJET

Photo slideshow

SARAH LACHIHEB

5 Mai 2019

Table des matières

1 Introduction

Le but de ce projet est d'obtenir un "photo slideshow", c'est à dire une présentation composée de slides dans un ordre de passage optimal selon un certain score. En effet, l'ordre des photos est régit par une notation, ce qui permet de les mettre dans le meilleur ordre possible selon cette notation.
On sait aussi qu'il y a deux types de photos, celles horizontales qui sont une par slide, et les photos verticales qui doivent être deux. Aussi, on sait que chaque photo dispose d'une liste de mots clefs.
Le score d'une transition entre deux vignettes successives S_i et S_{i+1} est définit par le minimum entre :

- ➢ Le nombre de mots clefs commun entre les deux vignettes.
- ➢ Le nombre de mots clefs dans S_i et non dans S_{i+1}.
- ➢ Et l'inverse, soit le nombre de mots clefs dans S_{i+1} et non dans S_i.

Dans l'exemple ci dessous en Figure 1, on voit que la slide i possède 5 mots clefs dont 2 en commun avec $i+1$, et $i+1$ en possède 4. Ainsi, le score est $min(2,(5-2),(4-2)) = min(2,3,2)$ donc le score est de 2.

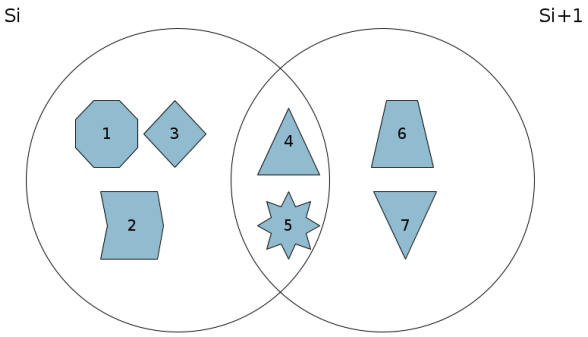

FIGURE 1 – Exemple de score entre deux vignettes.

Le langage de programmation utilisé pour ce projet est le C++, il semble être un bon choix car nous avons besoin de rapidité étant donné la taille des instances. Pour résoudre le programme linéaire de l'exercice 5 c'est Cplex qui a été utilisé.

2 Un problème NP-Complet

Un problème de la classe $\mathscr{NP}$ est un problème de décision qui peut être décidé par une machine de Turing non déterministe en temps polynomial par rapport à la taille de l'entrée. Cela revient à dire que l'on peut vérifier en temps polynomial si une solution candidate est bien solution. Un problème est dit $\mathscr{NP}$-Complet, c'est à dire un problème complet pour la classe $\mathscr{NP}$ si c'est un problème qui vérifie les deux conditions suivantes :

- ➢ Il est possible de vérifier une solution en un temps polynomial (problème $\mathscr{NP}$).
- ➢ Tous les problèmes de la classe $\mathscr{NP}$ se ramènent à celui-ci via une réduction polynomiale, cela signifie que le problème est au moins aussi difficile que tous les autres problèmes de la classe $\mathscr{NP}$.

Si on réduit le problème à n'avoir que des photos horizontales, le problèmes est déjà NP-Complet. En effet, on va démontrer que l'on peut réduire ce problème au problème NP-Complet du Voyageur de Commerce version décision :

- ➢ Le problème du voyageur de commerce, appelons le problème $\mathscr{A}$, peut se définir de cette manière :
  Entrées : Un graphe complet $G = (V,E)$ avec V le nombre de sommets du graphes, soit le nombre de villes, D la matrice de distances entre ces villes et un entier k.
  Question : Est ce qu'il existe un circuit passant au moins une fois par chaque ville et dont la somme des coûts des arêtes est au plus k ?
- ➢ On doit montrer que $\mathscr{A} \in \mathscr{NP}$.
  En effet, si une solution nous est donnée on peut facilement vérifier dans le temps d'un parcours de

liste, soit en temps linéaire, s'il n'y a pas de doublons et si la distance entre ces villes est bien inférieur à k (ceci ce fait à l'aide d'une simple somme).

- ➢ Notre problème, appelons le problème $\mathscr{B}$, peut se définir de cette manière :
  Entrées : Un graphe complet $G = (V, E)$ avec V le nombre d'images correspondant aux sommets du graphes et D la matrice de scores entre ces images et un entier b.
  Question : Existe t-il une chaîne passant par toutes les images dont le score est plus grand que b?
- ➢ On peut réduire ce problème en définissant l'ensemble des vignettes comme chacune des villes de l'ensemble V, plus une villes factice v_f. Les distances entre ces villes sont données par $d'_{ij} = -d_{ij}$ où d_{ij} est le min des trois scores à calculer pour deux vignettes successives, et toutes les distances entre la vignette factice et une autre vignette est de 0. Et en effet même si on cherche à maximiser le score, cela est identique à minimiser l'inverse des scores.
  Ainsi cette transformation est bien polynomiale et fait de notre problème un problème NP-Complet car il correspond au problème décisionnel du Voyageur de Commerce.
- ➢ Une réduction de $\mathscr{B}$ à $\mathscr{A}$ est un algorithme en temps polynomial transformant toute instance de $\mathscr{B}$ en une instance de $\mathscr{A}$. Ainsi, si un algorithme résout $\mathscr{A}$, on sait aussi résoudre $\mathscr{B}$ et $\mathscr{B}$ est donc au moins aussi difficile à résoudre que $\mathscr{A}$.

3 Description des différentes méthodes

La première méthode à effectuer est de ranger dans l'ordre de lecture les différentes images, elle nous permettra une comparaison avec les différentes méthodes utilisées et cette section détaille leurs implémentations.

3.1 Méthode gloutonne

L'algorithme glouton proposé par l'exercice 3, consiste à choisir un point de départ aléatoire puis à prendre la meilleure image à suivre, en comblant de la meilleur manière si c'est une image verticale. Cet algorithme n'est pas utilisable sur des instances de grandes tailles.
L'idée du code reste la même, c'est pour chaque vignette on cherche dans toutes les autres vignettes restantes la meilleure solution. Dans le but de le rendre plus rapide, j'ai abordé plusieurs solutions qui ont pour but de réduire la durée de la recherche de la meilleure combinaison, comme comparer que les m vignettes suivantes, choisir un pourcentage x de vignettes choisi aléatoirement mais aussi, si on ne s'améliore pas pendant n itérations alors on garde le meilleur résultat trouvé jusqu'alors. Cependant les résultats de ces méthodes furent décevant.

L'objectif était de trouver un moyen de "couper" dans les vignettes à choisir.

Dans un premier temps, on estime que si le score avec l'image courante est supérieure ou égale au nombre de mots clefs de la vignette précédente divisé par un nombre n alors on peut s'arrêter. En effet, comme le score est un min entre le nombre de mots clefs commun aux deux listes et ceux non dans la liste, pour chacune des deux listes, le maximum possible est le nombre de mots clefs de la vignettes ayant le moins de mots clefs divisé par 2. On peut donc imposer un arrêt(avec $break$) si un score souhaité est atteint, dans le but de réduire le temps de recherche de la meilleure vignette successive.

La meilleure option trouvée a été d'imposer un seuil au delà duquel on s'arrête, cette solution permet aussi d'avoir un levier sur la rapidité du calcul de la meilleure présentation. En effet, si la division par deux me permet d'avoir un meilleur score qu'une division par 3 ou 4, s'il ne peut être atteint alors cela revient à faire le glouton de la question 1 de l'exercice. Ainsi, si la présentation n'est pas assez rapide à se former, on peut choisir une division par 3 ou même par 4 de la taille de la liste.

On peut voir ci dessous l'implémentation de l'algorithme avec le nombre seuil nb qui est à donné en argument et qui nous permet de moduler la difficulté de chercher une vignette suivante.
L'algorithme est ici simplifié mais si une vignette Verticale est choisie comme meilleure option alors l'algorithme cherche la seconde vignette Verticale qui maximise la transition suivant ce même principe.

Algorithm 1 Glouton

Input ↓ :

$data$ ▷ les données lues dans le fichier

nb ▷ permet de définir le seuil d'acceptation d'une vignette candidate suivante

Output ↑ :

$L_{presentation}$ ▷ la présentation optimale trouvée par le glouton

```
1:  L_v ← listeVignettes(data)
2:  L_score ← []
3:  L_presentation ← []
4:  L_presentation[0] ← firstVignetteRandom(L_v)
5:  L_v ← remove(L_presentation[0])
6:  cpt ← 0
7:  while cpt != sizeof presentation do
8:    for v_j ∈ L_v do
9:      L_score[j] ← calculateScore(L_presentation[cpt].getKeyWords(), v_j.getKeyWords())
10:     if ! (L_score[j] < L_presentation[cpt].getKeyWords()/nb) then
          BREAK
11:     end if
12:   end for
13:   cpt ← incrementation(1)
14: end while
15: return L_presentation
```

3.2 Méthode méta-heuristique

La difficulté de cette méthode est d'obtenir un voisinage efficace, c'est à dire qui confère des opportunité d'améliorations et étant donné la proximité de ce problème avec le problème du voyageur de commerce, on peut naturellement se tourner vers un voisinages k-opt. En effet l'heuristique de Lin-Kernighan, dérivée du k-opt, est connu pour être une heuristique très efficace. Pour insérer beaucoup de diversité, c'est un 3-opt qui a été implémenté dans un premier temps mais devant le grand nombre d'alternatives (factorielle) et l'amélioration peu importante, ainsi c'est un algorithme génétique basé sur l'implémentation demandé qui a été choisie pour améliorer la méta-heuristique proposée.

Ainsi, une fois que l'on a la présentation proposée par l'algorithme glouton, on fait un nombre n de fils auquel on applique une mutation. La mutation ici est celle proposée dans l'exercice, c'est à dire que selon un nombre aléatoire, soit on change deux vignettes, soit on interverti deux images verticales dans deux vignettes différentes.
Ensuite on applique cette transformation à p générations. Le nombre de fils n et de générations p dépendent du temps que l'on souhaite accorder à l'amélioration de la solution retournée par l'algorithme glouton mais pas de la taille de l'instance (une transitions si l'image est la première ou la dernière sinon deux transitions par échange). En effet le calcul du score ne s'effectue que sur les transitions touchées par l'échange et leur nombre est toujours le même peu importe qu'il y ait 1 000 ou 90 000 images. Cela permet à la méta-heuristique de ce détacher de la dimension bien que plus un fichier est grand plus il a besoin de fils et de générations. L'adaptabilité est un point positif aux algorithmes génétiques.

Algorithm 2 Métha-heuristique génétique

Input ↓ :
S ▷ solution retournée par l'algorithme Glouton
n ▷ nombre de fils dans l'algorithme génétique
p ▷ nombre de génération de l'algorithme génétique
Output ↑ :
$L_{presentation}$ ▷ la présentation optimale trouvée par cet algorithme

```
while cpt < p do
    L_sons ← createNsons(S)
    S ← evaluationsSonsAndTakeBest(L_sons)
    cpt ← incrementation(1)
end while
L_presentation ← S
return L_presentation
```

3.3 Méthode Exacte

Il y a une forte ressemblance entre le problème du TSP et celui qui est étudié dans ce projet. En effet, les buts sont similaires, soit trouver une suite de villes, respectivement vignettes, qui minimise, respec. maximise, un score. Cependant, il y a plusieurs différences, notamment sur la direction de l'optimisation car ici c'est une maximisation d'un score et non une minimisation(de la distance), mais aussi la maximisation s'effectue sur un minimum entre trois scores dans notre cas. Aussi et certainement la plus grande distinction entre les deux c'est que notre problème ne consiste pas en la réalisation d'une cycle comme dans le TSP.
Avec l'objectif d'utiliser des programmes linéaires proche du TSP, on créé une ville factice à une distance de 0 de toutes les autres villes. Ensuite lorsque l'on cherche à former le tour optimal, donc l'ordre des vignettes retourné par Cplex, on positionne en première position la ville factice comme dans l'exemple en Figure 2 ci dessous où A est la vignette factice et où l'ordre désigné est $A-B-E-C-D-A$. Ainsi, une fois que l'on supprime la vignette A, la valeur de la fonction objective ne change pas puisque les scores entre (A, B) et (D, A) sont de 0 et donc l'ordre optimal retournée serait ($B-E-C-D$) et il ne contiendrait donc pas de cycle car le cycle formé par le résultat du programme linéaire du TSP serait brisé par la suppression de la "ville" (vignette) factice.

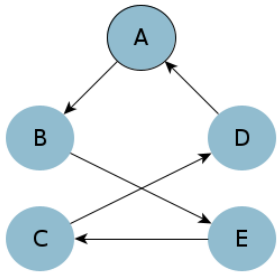

FIGURE 2 – Exemple avec une vignette factice A.

On suppose qu'une arête est évaluée par un vecteur à trois indices c où $c = (c^1, c^2, c^3)$ et où chacune des indices est une évaluation d'un score. Ainsi, en utilisant le programme linéaire éliminant les sous tours par les flots on peut en déduire le programme suivant :
Données : $n+1$ villes avec c_{ij}^k le vecteur coût du passage de la vignette i à la vignette j.
Objectif : Déterminer un circuit (qui sera ensuite brisé sans incidence sur la valeur de la fonction objectif grâce à la vignette factice) entre les vignettes de score maximal.
Variables : On note $x_{ij} = 1$ si la vignette i est suivit de la vignette j et 0 sinon.

$$
\begin{array}{lll}
\max & \min_k & \sum_{i,j}\sum_k c_{ij}^k x_{ij} \\
s.c & \sum_{j\in\mathcal{V}} x_{ij} = 1 & \forall i \in \mathcal{V} \\
 & \sum_{i\in\mathcal{V}} x_{ij} = 1 & \forall j \in \mathcal{V} \\
 & \text{<Élimination des sous tours par la formulation des flots>} & \\
 & x_{ij} \in \{0,1\} &
\end{array}
$$

Cependant, le max d'un min n'est pas linéaire, on va donc utiliser une variable auxiliaire, pour sa linéarisation, soit u.

$$
\begin{array}{lll}
\max & u & \\
s.c & u \leq \sum_{i,j} c_{ij}^1 x_{ij} & \\
 & u \leq \sum_{i,j} c_{ij}^2 x_{ij} & \\
 & u \leq \sum_{i,j} c_{ij}^3 x_{ij} & \\
 & \sum_{j\in\mathcal{V}} x_{ij} = 1 & \forall i \in \mathcal{V} \\
 & \sum_{i\in\mathcal{V}} x_{ij} = 1 & \forall j \in \mathcal{V} \\
 & \text{<Élimination des sous tours par la formulation des flots>} & \\
 & x_{ij} \in \{0,1\} &
\end{array}
$$

Aussi, il semble très probable que ce programme linéaire ne soit pas un bon outil pour la résolution de ce problème, en effet on peut imaginer la difficulté de résoudre un TSP de taille de 8000 villes, ce qui ne représente qu'un pourcent du fichier sur lequel nous devons utiliser ce programme linéaire.

3.4 Méthode d'heuristique d'arrondis

La première méthode d'arrondi proposée consiste à ne pas forcer la valeur des x_{ij} à 1 ou 0. Ensuite on range de manière croissante les valeurs de ces x_{ij} dans le but de former une présentation si cela ne présente pas de cycle. Cela retourne donc une solution moins performante que celle exacte.
Une amélioration a été de tenter de former une présentation même si lorsque l'on range la présentation par ordre croissant cela conduit à un cycle. Pour cela il suffit de ne pas permettre la formation de cycle en retirant les vignettes qui ont été ajoutées à la présentation. Ainsi l'ordre reste possible.

Aussi il est connu que cette méthode gloutonne ne soit pas optimale par expérimentation, ainsi pour tenter d'améliorer cette solution, on peut lui ajouter la méta-heuristique basé sur un algorithme évolutionnaire.

On pourrait rendre ce PLNE plus performant en utilisant des formulations non compacte et donc en utilisant des algorithmes tels que le Branch-and-Cut.

4 Analyse pratique et théorique

Les résultats présentés ci-dessous sont issue de moyennes sur les 10 meilleurs instances pour les algorithmes ayant des résultats et des temps d'exécution fluctuants(dons pas la méthode exacte qui ne nécessite qu'une seule exécution), avec un arrondi sans chiffre après la virgule. Les algorithmes, glouton et méta-heuristique, sont testés sur tous les fichiers mais les algorithmes exacte ne sont développés que pour le fichier "b_lovely_landscapes.txt" qui ne contient que des photos Horizontales.

4.1 Fichier a_example.txt

Algorithme	Pourcentage	Temps en s	Score
Ordre de Lecture	100	1.5e-05	2
Glouton	100	5.1e-05	2
Méthaheuristique	100	2	2

TABLE 1 – Comparaison pour le fichier "a_example.txt"

4.2 Fichier b_lovely_landscapes.txt

Ce fichier possède 80 000 images, toutes de type Horizontales. Le paramètre de la méthode glouton est très utile dans le cas de ce fichier, en effet ici on constate que les scores sont soient de 0 soient de 3 et donc le seuil imposé par notre algorithme doit être grandement réduit car les listes de mots clefs sont longues et donc cela reviendrait à faire un glouton classique. Bien que n=2 soit en dure dans le code, pour ce fichier il faut le modifier afin d'avoir ces résultats. Ici on prend n=8, on pourrait aussi placer le seuil à 3.
Aussi la méta-heuristique n'a pas été poussée car on voit qu'elle ne permet pas d'amélioration masi qu'elle prend beaucoup de temps. Cependant elle se révèle efficace pour les autres fichiers.

Algorithme	Pourcentage	Temps en s	Score
Glouton	0.5	0.78	12
Méta-heuristique	0.5	172.83	12
Exacte	0.5	2.59	12
Heuristique d'arrondi	0.5	1.98	12
Glouton	2	12.91	252
Méta-heuristique	2	676.85	252
Exacte	2	5300.93	255
Heuristique d'arrondi	2	26.77	240
Glouton	5	85.39	1 374
Méta-heuristique	5	3 108.69	1 374
Heuristique d'arrondi	5	1 542.16	1 329
Glouton	10	305.39	5 298
Méta-heuristique	10	7 163.17	5 298
Glouton	50	4 022.71	79 617
Ordre de Lecture	100	0.23	12
Glouton	100	11 356.4	204 468

TABLE 2 – Comparaison pour le fichier "b_lovely_landscapes.txt"

4.3 Fichier c_memorable_moments.txt

Le fichier possède 100 images des deux types mélangés. Ici, on a p le nombre de générations, n le nombre de fils pour la méta-heuristique et n pour le glouton signifie par combien on divise la taille de la vignette précédente pour avoir un seuil d'arrêt. Comme la taille du fichier est petite, 100 images, on peut utiliser l'algorithme avec n=2.
Pour la méta-heuristique, p = 100 n = 1000.

Algorithme	Pourcentage	Temps en s	Score
Glouton n=2	50	0.68	724
Méta-heuristique	50	155.12	727
Ordre de Lecture	100	0.0035	152
Glouton n=2	100	2.37	1592
Méta-heuristique	100	240.31	1601

TABLE 3 – Comparaison pour le fichier "c_memorable_moments.txt"

4.4 Fichier d_pet_pictures.txt

Le fichier possède 90 000 images des deux types mélangés. Ici, étant donné la taille du fichier, on test le glouton avec n=2 et n=3. Pour la Méta-heuristique, p = 100 n = 1000.

Algorithme	Pourcentage	Temps en s	Score
Glouton n=2	50	949.09	166 971
Glouton n=3	50	229.28	136 846
Méta-heuristique	50	11 638.9	171 550
Ordre de Lecture	100	0.28	190 961
Glouton n=2	100	3 005.10	335 840
Glouton n=3	100	904.91	272 217
Méta-heuristique	100	94 896.70	350 622

TABLE 4 – Comparaison pour le fichier "d_pet_pictures.txt"

4.5 Fichier e_shiny_selfies.txt

Le fichier possède 80 000 images exclusivement de type vertical. La méta-heuristique se révèle ici très longue, pour un fichier exclusivement remplie de photos verticales.

Algorithme	Pourcentage	Temps en s	Score
Glouton n=2	10	654.65	37 503
Glouton n=3	10	236.18	24 186
Méta-heuristique	10	3 590.05	37 504
Glouton n=2	20	1 887.97	78 293
Glouton n=3	20	560.04	66 398
Méta-heuristique	20	9 734.38	66 404
Glouton n=3	50	1 495.97	158 094
Méta-heuristique	50	123 368.98	158 144
Ordre de Lecture	100	0.33	114 685
Glouton n=2	100	27 556.97	421 803
Glouton n=3	100	5 556.97	324 066

TABLE 5 – Comparaison pour le fichier "e_shiny_selfies.txt"

4.6 Analyse des résultats

Le fait d'afficher les scores de la présentation formée dans l'ordre de la lecture du fichier nous permet de nous rendre compte du rangement du fichier. On peut voir ici que le fichier "d_pet_pictures.txt" qui possède 90 000 images est mieux rangé que "e_shiny_selfies.txt" bien qu'il est 10 000 images de moins. Aussi on peut remarquer que "b_lovely_landscapes.txt" est de loin le plus mal rangé car bien qu'il possède 80 000 images, son rangement par ordre de lecture n'obtient qu'un petit score de 12.
Ces score nous permettrons de mieux mettre en reliefs les temps de nos différents algorithmes.

Les résultats suivent ce qui pourrait être logique du point de vue théorique. En effet, l'algorithme dit "Glouton amélioré" reste le plus rapide mais on peut voir que ses résultats restent améliorables. Dans un premier temps on voit de part les améliorations faites avec la méta-heuristique que l'on applique à la présentation retournée par ce dernier. Aussi on peut le voir lors de la comparaison, sur un plus faible pourcentage, sur fichier "b_lovely_landscapes.txt" auquel on peut appliquer une méthode exacte, et où l'on constate de la progression qui pourrait être faite.
Cependant, étant donnée la taille des instances, 80 000 ou encore 90 000, ou de leur composition, car le PLNE n'est applicable que sur le fichier "b_lovely_landscapes.txt", le glouton ayant subi plusieurs améliorations est une très bonne option. La méta-heuristque est très utile sur les instances de plus petites tailles mais devient vite très gourmande en temps lorsque la taille de l'instance augmente fortement.
La relaxation du PLNE le rend beaucoup plus rapide mais il perd de son efficacité.

5 Améliorations et meilleurs résultats

Étant donné les résultats, j'ai souhaité apporter des améliorations à l'algorithme glouton. C'est le plus rapide donc on dispose de plus de possibilité pour l'améliorer.
Dans un premier temps j'ai pensé que le traitement des photos verticales en cours d'élaboration de la présentation pouvait être améliorable et j'ai donc implémenté une méthode pour former des couples de photos verticales avant de créer la présentation. Deux photos verticales formaient un binôme lorsque leur nombre de mots en commun est maximal. Cette idée peut sembler logique, en effet on place deux photos côte à côte si elle se ressemble beaucoup en terme de mots clefs.
Cependant, l'idée de l'élaboration des binômes de photos verticales avant l'élaboration de la présentation retourne des résultats bien moins bon et cette voie n'a donc pas été développée.

La principale idée était de chercher un moyen d'éviter de calculer le score, puisque c'est cela qui prend le plus de temps, de certaines vignettes dont on pouvait supposer savoir à l'avance qu'elle ne pouvait être un choix. Pour cela on garde à chaque itération de recherche d'une meilleure vignettes suivante ou binôme et si ce score est supérieur ou égale à la moitié de la liste de mots clef suivants alors on ne calcul pas ce score. Cela peut nous faire perdre des points de score car ce n'est pas une vérité mais cette supposition nous permet d'en éliminer beaucoup.
Bien que la première vignette reste aléatoirement déterminée, si elle est de type Verticale, comme cela est forcément le cas dans le fichier "e_shiny_selfies.txt" , alors on ne place pas la seconde photo binôme de manière aléatoire. En effet, on choisie la première qui répond positivement à une test qui détermine si ces deux photos ont au moins un mots clef en commun. Ainsi, on s'assure que les deux photos binôme dans la première slide ne soit pas trop différentes.

Fichier	Score
a	2
b	204 573
c	1 621
d	357 054
e	422 269
$Total$	985 519

TABLE 6 – Meilleurs résultats pour chacun des fichiers

6 Renseignements préalables sur l'exécution du code source

L'exécution du code se fait via la commande à faire sur le terminal. En effet, après avoir effectué la commande *make* il faut exécuter l'exécutable avec 3 arguments suivant, en premier le numéro de la méthode souhaitée (0 pour le sens de lecture, 1 pour le glouton, 2 pour la méta-heuristique, 3 pour le PLNE et 4 pour l'heuristique d'arrondi).
Cela retourne, si nous faisons une erreur :

Veuillez saisir : **./exec <Numéro de méthode souhaitée> <nom ficher(.txt)> <pourcentage>** Les numéros de méthodes associés sont :
0 : méthode sens de lecture du fichier.
1 : méthode gloutonne.
2 : méthode métha-heuristique.
3 : méthode exacte PLNE
4 : méthode méthode heuristique d'arrondi PL

7 Conclusion

Ce projet a une originalité qui force à creuser différentes méthodes pour les adaptés aux différents "problèmes dans le problème". En effet, la gestion des fichiers *b* et *e*, soient ceux qu'avec des images verticales et ceux qu'avec des horizontales doivent bénéficier d'une même méthode. Aussi les fichier *c* et *d* qui sont de la même composition ont une taille qui force encore l'adaptation de notre algorithme, passant de 1 000 à 90 000.
Ce projet m'a permis de comprendre comment adapter différentes méthodes et combinaisons pour parvenir à avoir de bons résultats sans la certification d'une méthode exacte. Aussi j'ai pu comprendre l'importance des algorithmes gloutons, et des méthode heuristiques face à ma confrontation sur des problèmes de très grande taille.