Rapport Projet Algorithme et Complexité
Paul Decanter & Guillaume Roth

Note :
Lors de l'execution du programme, des fonctions de test sont disponibles pour chaque question du projet que nous avons traitée.

2.2/ Calcul du fingerprint
Q5/
Nous avons testé le programme sur les fichiers test1 et test4 avec de grandes valeurs de p et avons obtenu les affichages suivants
fingerprint du fichier test1 avec p = 5279233 : 4544154
fingerprint du fichier test4 avec p = 5279233 : 3028786
fingerprint du fichier test1 avec p = 65633 : 12553
fingerprint du fichier test4 avec p = 65633 : 15457
fingerprint du fichier test1 avec p = 5926729 : 4919480
fingerprint du fichier test4 avec p = 5926729 : 3743975
fingerprint du fichier test1 avec p = 3443603 : 3090958
fingerprint du fichier test4 avec p = 3443603 : 1250814
fingerprint du fichier test1 avec p = 8124931 : 4548898
fingerprint du fichier test4 avec p = 8124931 : 3163357
fingerprint du fichier test1 avec p = 767677 : 246830
fingerprint du fichier test4 avec p = 767677 : 423854

Nous pouvons constater que les fingerprint sont effectivemetn différents.

Q6/
Lorsque nous lançons notre programme sur les fichiers test3.xpm et test4, nous pouvons constater que leurs fingerprints sont très souvent identiques.

Q7/

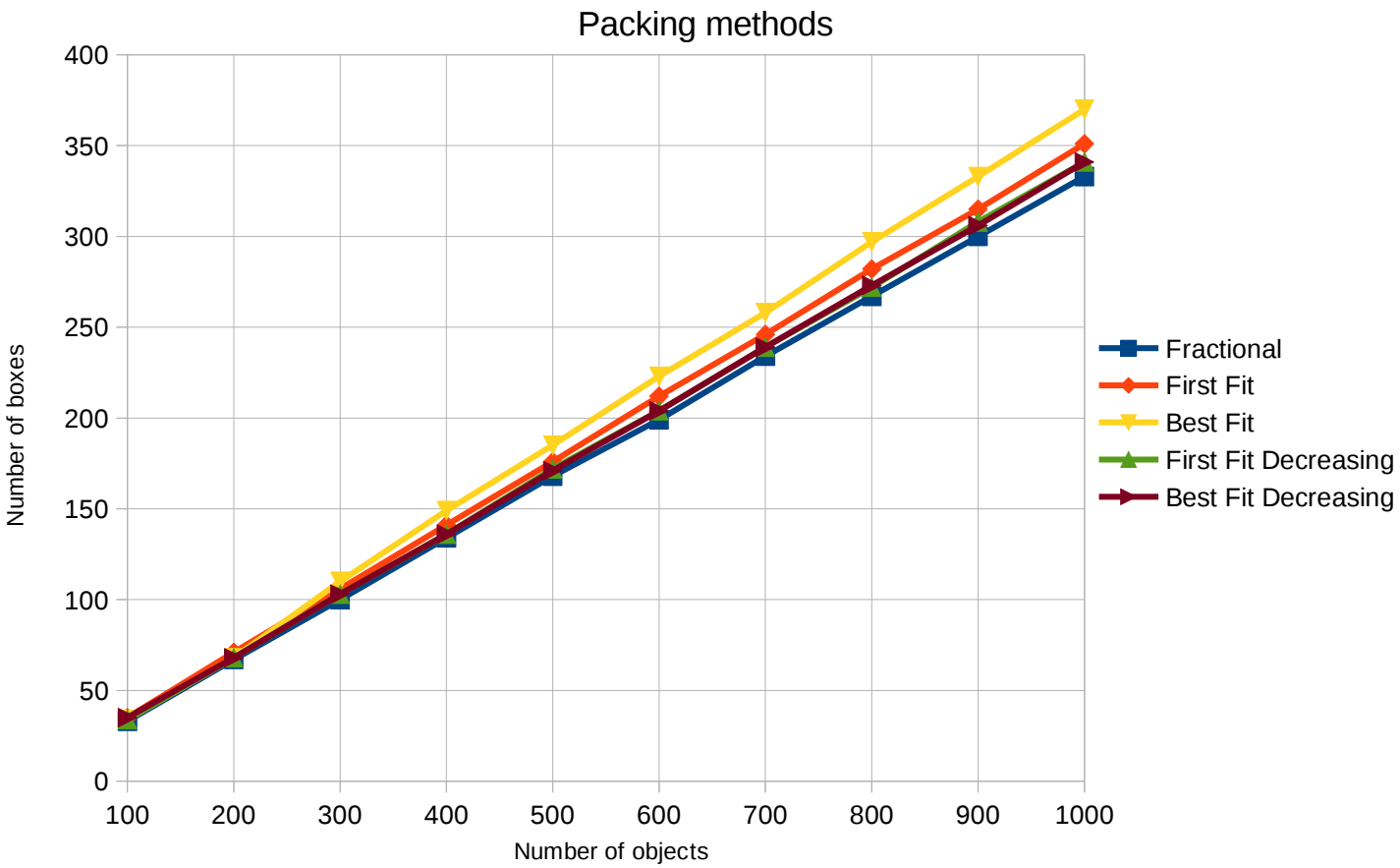
Q8/
Nous avons, grace a la Q5, pu constater que les fichiers test1 et test4 possèdent pratiquement systematiquement le meme fingerprint, quelque soit la valeur de p choisie.
Nous pouvons donc supposer que ces deux fichiers sont plus ou moins identiques.
Nous emettons donc l'hypothese que le fichier test4 (de type binary) a ete obtenu en convertissant le fichier test1 (de type image xpm) en fichier binary.

Q9/
Comme pour chaque projet effectue dans un laps de temps un peu limité avec une question bonus un peu compliquée : RIP le bonus... :'(

Q11/
Etant donne que les fichiers test2 et test4 ont respectivement une taille similaire aux fichiers test5.xpm et test3.xpm, nous avons decidé de tester ces fichiers.
Voici donc la trace d'execution du test correspondant :
-- testFingerContains --
test2 est contenu dans test5.xpm : false
test4 est contenu dans test3.xpm : true

3/ Problème de bin packing
Dans cette partie, chaque objet sera (au niveau du programme) represente par sa hauteur.

Q16/ Grâce à la commande de test numéro 12 du programme, nous avons généré un tableau à partir duquel nous avons été capables de créer le graphique suivant :



Ce graphique nous montre l'évolution du nombre de boites en fonction du nombre d'objets pour chaque méthode testée.
Comme nous pouvions nous y attendre, la méthode fractional packing est celle qui demande le moins de boites, étant donné qu'elle permet de découper les objets.
Nous pouvons également remarquer que la méthode best fit semble moins efficace que first fit, car elle semble moins efficace. Il semble que cette première porte mal son nom. En effet, comme présenté dans l'énoncé, celle-ci doit privilégier le choix d'une choix dans laquelle il restera le plus de place après y avoir déposé l'objet. Prenons un exemple : soit deux boites de taille 10. La boite 1 est remplie à valeur 8, et la boite 2 à valeur 4. Si nous prenons à présent un objet de taille 2, l'algorithme va comparer la place qu'il restera dans chaque boite après dépôt de l'objet, et va choisir la boite dans laquelle il restera le plus de place : la boite 2, alors qu'il aurait pu terminer de remplir totalement la boite 1. Il s'agit d'une erreur d'optimisation. Pour remédier à ceci, nous aurions pu ajouter une condition selon laquelle l'algorithme choisirait en priorité une boite que l'objet remplirait totalement avant les autres conditions citées ci-dessus.
Enfin, nous pouvons nous apercevoir que les méthodes first fit decreasing et best fit decreasing semblent plus efficaces que leurs versions "non decreasing". Cela s'explique par le fait que ces algorithmes trient les objets avant d'appliquer les méthodes du même nom. On remarquera que, pour les mêmes raisons que citées dans le paragraphe précédent, first fit decreasing est plus efficace que best fit decreasing.

Note : nous sommes conscients que notre programme génère un tableau d'entiers, ce qui est plutôt gênant pour un tableau de moyennes. Nous avons choisi de penser que cela reste acceptable, dans le sens où les données restent exploitables, car il s'agit simplement de comparer les résultats obtenus par les méthodes, et non d'étudier des chiffres précis.

A propos de notre programme :
La partie fingerprint est traitée par deux fichiers :
- MainFingerprint qui implémente les algorithmes et quelques fonctions annexes
- TestsFingerprint qui se charge d'éxécuter les fonctions permettant de répondre aux questions du sujet.
La partie bin packin est traitée par trois fichiers :
- MainBinPacking qui implémente les algorithmes et quelques fonctions annexes
- TestBinPacking qui se charge d'éxécuter les fonctions permettant de répondre aux questions du sujet.
- FenetreTableau auquel on envoie les données de la question 16 et qui va générer un tableau comparatif des méthodes suites aux tests, que l'on a pu utiliser par la suite pour produire le graphique en page 2.
Le programme se lance en éxécutant Launcher.

A propos de la répartition du travail :
Dans une tentative d'efficacité accrue, nous avons décidé de programmer à deux sur la même machine. Cela nous a en effet permis de résoudre plus rapidement certains problèmes posés par la mise en place de certains algorithmes.