Université de Lille 1 SCIENCES ET TECHNOLOGIES

Master 1 Informatique

TP ACT 4

Compte rendu de TP

Antoine CANDA
Theo VERSCHAEVE

Version 1.0
30 décembre 2016

Table des matières

Chapitre 1

Utilisation de l'application

Il y a différents mode possible, qui sont plus au moins fonctionnel.
Le premier est le mode aléatoire qui a deux versions qui sont tirage aléatoire (1) ou mélange aléatoire (2) :
java -jar act.jar -alea data.in <version>
avec version = 1 ou 2.
Le seconde mode est l'heuristique gloutonne :
java -jar act.jar -heur data.in <nbCycle>
avec nbCycle qui représente le nombre de boucle à effectuer.
Le troisième mode est la tentative d'amélioration qui n'a pas été concluante :
java -jar act.jar -ame data.in <sol.txt>
avec sol.txt représentant le certificat de la solution que l'on souhaite améliorer. Cette solution n'est pas fonctionnelle.
Le quatrième mode est similaire au mode 3 non fonctionnel toujours. Il est basé sur le mode 2.
java -jar act.jar -ame2 data.in

Chapitre 2

Certificat et validation

Question 1 :

Implémentation possible d'un certificat

Un certificat est le fichier de validation.
Il est composé de deux parties :
La première ligne du fichier correspond au nombre de part de la solution.
Les autres lignes correspondent aux informations d'une part, c'est à dire : ligne 1 colonne 1 ligne 2 colonne 2.

Question 2 :

Taille d'un certificat par rapport à l'entrée

L'entrée correspond à un tableau bidimensionnel de longueur L et largeur l, de taille L*l.
On y ajoute une ligne composé de 4 entiers représentant la longueur L, la largeur l, le nombre minimal de jambon par part et la surface maximale d'une part royal de pizza.
Pour une position tableau[L][l] on a un octet qui correspond à un caractère ASCII ("T" ou "H").
On a donc l'entrée qui fait une taille (L*l)* 1 octet + 4 * taille d'un entier (on peut supposer 4 octets pour un entier).
La représentation d'un entier dans notre cas se fait généralement sur 4 octets. Et on a 4 octets par part.
On a donc X (le nombre de parts) * 16 (le nombre d'octets). Une part est définie comme étant un rectangle de taille 3 à 12 dans notre cas (3 car minimum de morceaux de jambons présent et 12 car surface maximale d'une part).
On peut donc dire qu'on a au maximum (L*l)/3 parts et au minimum (L*l)/12 parts dans le cas où on a aucune perte. Si on considère que L*l = 10 800, alors on a entre 900 et 3600 parts de pizzas dans le cas optimale. Ainsi qu'un entier représentant le nombre de parts. Soit entre 900*16 + 4 (= 14 404 octets) et 3600*16 + 4 (= 57 604 octets) par certificats.
Vu que l'entrée a une taille d'environ 10 816 octets, on peut dire que le certificat a bien une taille polynomiale avec celle-ci.

Question 3 :

Implémentation d'un certificat

Voir code source.
La méthode ecrireFichier() créer le certificat à partir de la liste des parts.
Je me suis rendu compte tardivement que je n'avais pas créer de classe spécifique au certificat et que ça aurait pu être mieux fait.

Question 4 :

Implémentation d'un algorithme de validation de ce certificat et complexité

La méthode est possibleInsererPart(Part p).
L'algorithme de vérification du certificat valide la part elle-même (jambon et surface) et valide le fait qu'elle ne chevauche pas une autre part.
La validation de la part se fait en temps constant au niveau de la part (on considère que l'accès et la comparaison se font chacun en O(1)) donc pour toute les parts on arrive en temps linéaire (O(n)).
Pour vérifier que les parts sont disjointes une à une, on peut implémenter la pizza sous forme de tableau de booléen et tester qu'une part de pizza ne chevauche pas une autre part en la testant.
On arrive sur une vérification en O(n) avec n le nombre de case de la part. On a donc pour n part une validation en O(n carré).
On a donc une validation en O(n carré).
Le nombre de part de la pizza est basé sur la taille de la pizza en entrée : n est polynomiale par rapport à l'entrée et l'algorithme de vérification l'est donc aussi.

Question 5 :

Conclusion

On en déduit qu'on est en présence d'un problème NP car on a un certificat en taille polynomiale en fonction de l'entrée et un algorithme de vérification en temps polynomiale.
De même, le nombre part de pizza est limité par la taille de l'entrée et la couverture maximale d'un certificat peut au maximum représenté la surface totale de la pizza si il n'y aucune perte.

Chapitre 3

Solution aléatoire

Question 1 :

Génération de toutes les parts de pizza

Pour générer les parts, j'ai crée une méthode assez laide (getAllPart()) qui va parcourir l'entrée en totalité et crée toute les parts possibles valide dans les différentes formes qu'elle peut prendre (horizontal, verticale, carré etc...).
La génération des parts permet d'obtenir 105 536 part valides, c'est à dire de surface maximale 12, rectangulaire et avec 3 jambons minimum.

Question 2 :

Implémentation algorithme de base

Voir code source : méthode alea() et glouton() de Probleme qui tire aléatoirement une part ou mélange aléatoire la liste de toute les parts. On parcourt toute les parts possibles et on tente de les insérer à la solution.
En mélangeant aléatoirement la liste des parts ou en tirant aléatoirement une part dans la liste des parts, on peut créer des certificats valide. Mais les résultats ne sont pas très concluants : dans les 6000 en général.

Question 3 :

Optimale possible ?

En théorie, il est possible de tomber sur l'optimal si on itère un très grand nombre de fois cette solution.
Mais la probabilité est tellement faible qu'en pratique c'est quasiment impossible. Un test pratique me conforte dans l'idée parce qu'après plus de un demi million d'itérations, je n'ai pas eu de résultat vraiment supérieur au 6000.

Chapitre 4

Heuristique de pavage glouton

Question 1 :

Implémentation

En triant les parts par exemple par surface décroissante, ou coordonnées de part (notamment ligne), on peut améliorer le tirage. En faisant ça, j'ai pu obtenir des scores dans les 8500.
J'ai implémenter différentes méthodes de tri dans le package Part et la méthode testGlouton().
J'ai pu améliorer ces résultats en ajoutant un début aléatoire de ma solution et en modifiant les différentes méthodes de tris.
J'ai pu obtenir un score supérieur à 8900 ainsi.

Question 2 :

Autres méthodes possible ?

Différentes méthodes de tris ou parcours. J'ai crée différentes méthodes de tri (PartComparatorXXXX) dans le package Part.

Chapitre 5

Libre comme l'air

Je n'ai malheureusement pas réussi à faire cette partie malgré diverses tentatives avec beaucoup de petites méthodes utilitaire calculant des pourcentages de couvertures, des zones améliorables ou une boucle d'itération d'heuristiques gloutonnes avec conservation de la meilleure "bande de 12 lignes" des différentes méthodes et assemblages puis parcourt à nouveau des parts pour combler les manques présents...
J'ai essayé de retirer les parts avec le plus de jambon de la liste de solution puis trié les parts avec la surface minimale de jambon afin d'essayer d'obtenir une couverture plus efficace de la pizza sans succès... Je garde donc actuellement le score de 8959 sur le serveur (candaverschaeve).

Chapitre 6

Question bonus

Je n'ai pas implémenté la méthode basé sur l'independ set maximum malgré que je vois l'intérêt de cette méthode faute de temps.

Chapitre 7

Question super bonus

Pour prouver que le problème est NP-Dur il faudrait réduire un problème réputé NP-Dur (ou NP-Complet) dans le problème de pizza. On pourrait par exemple essayer de réduire une instance d'un problème de plus long chemin sans cycle dans un graphe dans notre instance de pizza.
Un cycle représenterait des parts de pizza qui sont incompatible entre elle (c'est à dire qu'elles se chevauchent). Les sommets représentent des parts de pizza valide (surface et nombre de jambon) et le plus long chemin sans cycle correspond au meilleur résultat possible.
Si le problème de la pizza est NP-Dur alors il est également NP-Complet car prouvé NP au début du TP.