École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. (33)2-47-36-14-14
Fax (33)2-47-36-14-22
www.polytech.univ-tours.fr

DI3

Rapport de projet S4

Projet tutoré 2: Sac à dos

Auteur(s)

Thomas Couchoud
[thomas.couchoud@etu.univ-tours.fr]
Victor Coleau
[victor.coleau@etu.univ-tours.fr]

Encadrant(s)

Yannick Kergosien
[yannick.kergosien@univ-tours.fr]
Polytech Tours
Département DI

Version du 20 janvier 2017

Table des matières

1 Information générales sur le projet **2**
- 1.1 Outils utilisés . . . 2
- 1.2 Structure du projet . . . 2

2 Fichiers **4**
- 2.1 Parser . . . 4
- 2.2 Instance . . . 4
- 2.3 Item . . . 5

3 Heuristique **6**

4 Metaheuristiques **7**
- 4.1 Local . . . 7
- 4.2 Tabou . . . 7
- 4.3 Genetique . . . 7

5 Autres **8**

Conclusion **9**

Annexes **9**

Introduction

Le projet que nous avons choisi est celui portant sur le problème du sac à dos (*Knapsack problem*). Celui-ci se généralise très simplement et donne lieu à de nombreux problèmes analogues.

Dans notre cas, nous devons remplir un sac à dos d'objets. Chaque objet a une certaine valeur prédéfinie ainsi que des "poids" dans différentes dimensions. On peut imaginer le cas où l'on tenterait de remplir sa valise pour partir en voyage. Chaque objet a une valeur selon l'importance qu'on lui donne ainsi que des "poids" qui pourraient être la place qu'il occupe, son poids réel etc.. L'idée ici est d'essayer de maximiser la valeur que nous emportons avec nous sachant que notre valise est limitée en poids et taille.

D'un point de vu mathématique, on peut modéliser ceci simplement :

- X Un vecteur définissant quels items sont dans le sac ou pas (ex : $(0; 0; 1)$ définira un sac avec seulement le 3ème item de pris).
- W_j Le poids maximum que le sac peut supporter dans la dimension i.
- $w_{i,j}$ Le poids du $i$ème item dans la $j$ème dimension.
- v_i La valeur du $i$ème item.

Les contraintes sont : $\forall j \in [0...m], \sum_{i=0}^{n} x_i w_{i,j} \leq W_j$
On appellera la fonction objectif $z(X)$ la fonction donnant la valeur d'un sac : $z(X) = \sum_{i=0}^{m} x_i v_i$

Chapitre 1

Information générales sur le projet

1.1 Outils utilisés

Afin de réaliser notre projet nous avons utilisé différents outils. Concernant les Systèmes d'exploitations nous avons utilisé Windows (Victor), OSX (Thomas) et Ubuntu (Travis CI). Les IDEs sont : CodeBlocks, CLion, Atom, Notepad++. Le compilateur utilisé est gcc, un Makefile est disponible pour la compilation.

1.2 Structure du projet

Le C ayant un langage rassemblant tous les fichiers en un lors de la compilation, il est nécessaire de choisir judicieusement ses noms de fonctions afin d'éviter les duplicatas. Dans notre cas nous avons choisi un formatage simple : [Nom du .c / Nom de la structure]_[Nom de la fonction]. Nous aurons donc des fonctions du type *population_ create(...)* ou bien *metaheuristicGenetic_ search(...)*.

Concernant l'organisation des fichiers en eux même, chaque type d'entre eux est localisé à un endroit différent. En effet nous avions commencé par mettre tous nos .c et .h dans un même dossier. Cependant, le projet grandissant assez vite, il a rapidement arrivé un stade où l'on se perd. Pour cela nous avons décidé de séparer les .h des .c puisque nous travaillons principalement sur les .c. Cela permet de s'y retrouver plus aisément. Ainsi la structure de notre dossier source est la suivante :

- src ⟶ Le dossier racine contenant nos .c pour le programme
  - headers ⟶ Le dossier contenant nos headers pour le programme
  - unit ⟶ Le dossier contenant nos .c pour les tests unitaires
    - headers ⟶ Le dossier contenant nos headers pour les tests unitaires

Intéressons-nous au dossier src, son contenu est :

- Parser ⟶ Regroupant les différentes fonctions afin de lire un fichier.
- Instance ⟶ Représentant une instance.
- Item ⟶ Représentant un élément de l'instance.
- Bag ⟶ Représentant le contenu du sac pour une solution indirecte.
- SolutionDirect ⟶ Représentant une solution directe.
- SolutionIndirect ⟶ Représentant une solution indirecte.
- Solution ⟶ Représentant l'union d'une solution directe et indirecte.
- Heuristic ⟶ Regroupant les fonctions liées à la résolution grâce à une heuristique.
- Scheduler ⟶ Regroupant les fonctions liées aux différents algorithmes pour l'heuristique.
- MetaheuristiqueLocal ⟶ Regroupant les fonctions liées aux différents algorithmes pour la metaheuristique locale.

— MetaheuristiqueTabou ⟶ Regroupant les fonctions liées aux différents algorithmes pour la metaheuristique tabou et contenant la structure Tabou.
— MetaheuristiqueGenetic ⟶ Regroupant les fonctions liées aux différents algorithmes pour la metaheuristique génétique et contenant la structure Population.
— MetaheuristicKaguya ⟶ Regroupant les fonctions liées à notre metaheuristique personalisée (contenant les structures Clan, ClanMember et DNA).

Chapitre 2

Fichiers

2.1 Parser - **Codé à 100%, Testé à 99.9%, Fonctionne: Oui, Valgrind: OK**

L'une des premières parties que nous devions réaliser est le parser. Lors de cette dernière, un choix important a du se faire : lisons-nous toutes les instances d'un fichier d'un seul coup ou lisons les nous une par une ?

Nous avions initialement décidé de les lire toutes à la suite. En effet, ce choix était celui de la simplicité. Nous avons voulu commencer simple afin de pouvoir avancer sans attendre sur les autres tâches à faire. Puis rapidement nous avons implémenté la seconde méthode. Celle-ci nous paru plus adéquate pour nos utilisations car elle permet d'éviter une utilisation importante de la mémoire pour pas grand chose. Certes nous avons du créer une structure Parser qui sert principalement à conserver les informations de la dernière lecture, mais ce choix nous paru être le meilleur.

Afin de satisfaire la seconde methode, une structure Parser a été créee et a ces fonction associées :

- parser_create ⟶ Permet de créer cette structure à partir du chemin d'un fichier.
- parser_destroy ⟶ Pour détruire la structure.
- parser_getNextInstance ⟶ Renvoi la prochaine instance du fichier ou NULL si l'on a atteint la fin.

A coté de cela des fonction génériques sont présentes :

- parser_readAllFile ⟶ Renvoi toutes les instances d'un fichier.
- parser_readInstance ⟶ Renvoi une instance à partir d'un fichier ouvert à la bonne position.
- parser_readLine ⟶ Lis la prochaine ligne non vide, ou renvoi NULL si on a atteint la fin du fichier.
- parser_lineToIntArray ⟶ Convertis un string composé de nombres séparés par tabulation en un tableau d'entiers.
- getLine ⟶ Lis la prochaine ligne du ficher.

2.2 Instance - **Codé à 100%, Testé à 99.9%, Fonctionne: Oui, Valgrind: OK**

Le fichier Instance comporte une structure nomée Instance qui contient ces propripétées suivantes :

- itemsCount ⟶ Représentant le nombre d'items dans l'instance.
- dimensionsNumber ⟶ Représentant le nombre de dimensions dans l'instance.
- items ⟶ Un tableau d'Item (section 2.3) étant les éléments de l'instance.

— maxWeights ⟶ Un tableau d'entier représentant le poids maximum pour chaque dimension.

Les fonctions suivantes agissent toutes à partir d'une instance :
— instance_initialize ⟶ Permettant de créer une instance sur le tas. La fonction instance_setMaxWeights devra par la suite être appelé. Le tableau d'item est créé mais chaque item devra être initialisé grâce à item_setWeight.
— instance_getItem ⟶ Permet de récupérer une item à un index précis dans l'instance.
— instance_setMaxWeights ⟶ Permet de définir le tableau des poids maximums de l'instance. Le tableau doit être alloué sur le tas.
— instance_getMaxWeight ⟶ Permet de récupérer le poids maximum sur une dimension précise.
— instance_destroy ⟶ Détruit une instance précédemment créée par instance_initialize.
— instance_item_getWeight ⟶ Récupère le poids de l'item à un certain index dans l'instance.
— instance_item_getValue ⟶ Récupère la valeur de l'item à un certain index dans l'instance.

2.3 Item - **Codé à 100%, Testé à 99.9%, Fonctionne: Oui, Valgrind: OK**

Le fichier Item contient une structure Item ayant pour propriétées :
— value ⟶ La valeur d'un item.
— weights ⟶ Un tableau de ses différents poids sur chaque dimension.

Les fonctions suivantes s'appliquent à partir d'une structure Item :
— item_initialize ⟶ Afin de créer un Item sur le tas.
— item_setWeight ⟶ Pour définir le poid d'un item dans la dimension souhaitée.
— item_getWeight ⟶ Pour obtenir le poid d'un item dans la dimension souhaitée.
— item_destroy ⟶ Afin de détruire un Item précédemment créé par item_initialize.

2.4 Bag - **Codé à 100%, Testé à 99.9%, Fonctionne: Oui, Valgrind: OK**

Le fichier Bag contient une structure Bag permettant de stocker les indices des items pris dans notre sac. Ses propriétées sont :
— bag_create ⟶ Permet de créer un bag sur le tas à partir d'une instance.
— bag_destroy ⟶ Permet de détruire un bag précédemment créé par bag_create.
— bag_appendItem ⟶ Ajoute un item dans le sac.
— bag_canContain ⟶ Permet de savoir si un item va pouvoir rentrer dans le sac.
— bag_getItemIndex ⟶ Permet de récupérer l'indice de l'item à un idex donné dans le sac.
— bag_getWeight ⟶ Récupère le poids actuel du sac dans la dimension demandé.
— bag_addWeight ⟶ Ajoute du poids dans le sac dans la dimension donnée.
— bag_saveItems ⟵ Ecris le bag dans un fichier.
— bag_print ⟶ Affiche le bag dans la console.

— bag_getCriticDimension ⟶ Renvoi l'index de la dimension critique.
— bag_toSolutionDirect ⟶ Permet de convertir un bag en une solutionDirect (section).
— bag_duplicate ⟶ Permet de dupliquer un bag sur le tas.

Chapitre 3

Heuristique

Dans le cadre des heuristiques, nous avons du implémenter nos propres critères de sélection. Nous allons ici vous en présenter deux.

Le premier se base sur l'algorithme de la dimension critique mais prend cette fois-ci en compte toutes les dimensions. Pour cela nous calculons pour l'item à l'index i un ratio qui est $r_i = \sum_{j=0}^{m} \frac{w_j}{W_j}$. Ce ratio sert par la suite à calculer un score temporaire afin d'appliquer l'heuristique $score_i = \frac{v_i}{r_i}$. De cette manière, plus l'item remplira le sac, plus le diviseur sera important et par conséquent, l'item aura un score faible.

Le s

Chapitre 4

Metaheuristiques

4.1 Local

4.2 Tabou

4.3 Genetique

Chapitre 5

Autres

Conclusion

Projet tutoré 2: Sac à dos

Rapport de projet S4

Résumé : Projet ayant pour objectif la réalisation d'un algorithme cherchant des solutions au problème du sac à dos multidimentionel.

Mots clé : sac à dos, algorithme, C, heuristique, metaheuristique, parser, directe, indirecte

Abstract : Project which objective is to find solutions for the multidimentional Knapsack problem.

Keywords : backpack, Knapsack, algorithm, C, heurictic, metaheuristic, parser, direct, indirect

Auteur(s)

Thomas Couchoud
[thomas.couchoud@etu.univ-tours.fr]
Victor Coleau
[victor.coleau@etu.univ-tours.fr]

Encadrant(s)

Yannick Kergosien
[yannick.kergosien@univ-tours.fr]
Polytech Tours
Département DI

Ce document a été formaté selon le format EPUProjetPeiP.cls (N. Monmarché)

École Polytechnique de l'Université de Tours
64 Avenue Jean Portalis, 37200 Tours, France
http://www.polytech.univ-tours.fr