Projet d'Optimisation et Recherche Opérationnel

Chutzpa William YEUMO BARKWENDE

May 2022

1 Introduction

Dans le cadre du projet, j'ai implémenté une méthode de séparation et évaluation pour résoudre le problème du sac à dos. Le principe du problème est simple : pour un poids limite, dans le cas d'un sac à dos, on a une liste d'objets donnés avec des valeurs différentes, et on cherche la combinaison d'objets optimale pour avoir une valeur maximale dans le sac. Ce programme respecte la contrainte d'intégrité et donc si on relâche cette contrainte.
C'est un problème très connu parmi les problèmes NP-complets. On peut retrouver le problème du sac à dos dans de nombreux domaines comme la cryptographies (dans les algorithmes de chiffrement asymétrique des années 70), dans le système financiers pour déterminer quels projets choisir dans des contraintes de temps au lieu de poids, etc...

2 Utilisation du programme

Afin d'utiliser le programme, il est nécessaire d'avoir un compilateur clang ou gcc. Si gcc est installé il suffit de lancer la compilation avec **make**. Sinon, il suffit de remplacer les $ (CC) par des $ (CL) dans le Makefile pour compiler avec clang. Ensuite pour lancer le programme on lance **./bin/main (T) (N) (P)** . T = le fichier texte à traité, N le nombre d'éléments, P le poids limite.

— (T) ici on entre le nom du fichier texte. Le format est simple on rentre, dans un ordre arbitraire le poids des différents objets. A savoir que si un objet existe en plusieurs exemplaires, il sera nécessaire de rentrer plusieurs son poids et sa valeur dans la ligne du dessous à dans la même colonne. Les poids par des espaces, de même pour les valeurs dans la ligne du dessous. Deux lignes sont nécessaires et suffisantes. Exemple, avec un problème avec 5 objets : Premier objet à un poids de 1 (unité arbitraire) et une valeur de 4 (unité arbitraire).

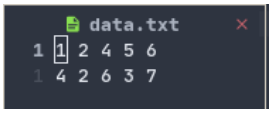


Figure 1 – Exemple de fichier texte

— (N) ici on entre le nombre d'objets à ranger dans le sac. Si on prends l'exemple précédent, on a 5 objets, 5 colonnes donc on entre 5.
— (P) ici on entre la limite de notre sac à dos. Si le poids maximal autorisé est de 12 (unité arbitraire) on entre 12.

Pour finir, on a besoin de traduire le résultat affiché par la console après l'exécution. J'affiche le parcours dans l'arbre en binaire. C'est à dire que les 0 représente l'ajout d'un objet à la couche où il se situe tandis que 1 signifie qu'on n'ajoute pas cet objet.

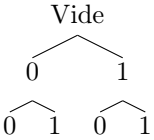
Si on prend le cas présenté dans le fichier txt avec 12 comme poids limite, la solution optimale a pour valeur 17 et le chemin optimale est : 0 1 0 1 0. Ce qui signifie qu'on prend le premier, le troisième et le cinquième objet mais pas le second et quatrième objet.
Le poids total du sac pour chaque combinaison est affichée, et comme le programme suit la restriction d'intégrité, le chemin optimale est celui qui atteint la valeur optimale pour le poids le plus bas.

3 Principe Mathématiques et Informatique

Pour notre problème on a un sac à dos à capacité limité. Mais on va utiliser une méthode qui va prendre toutes les possibilités puis retirer les cas non réalisables. D'une part on va calculer le poids de toutes les combinaisons de poids et valeurs possibles soit 2^N possibilités pour chaque variables, N étant le nombre d'objets. Pour calculer chacune de ses possibilités, on additionne les poids ou valeur des objets ajoutés dans chaque cas. Comme on considère dans l'ordre du fichier texte, l'ajout ou non d'un objet, et dans le code on travaille couche par couche, on obtient un temps de calcul très faible.

3.1 Séparation

La séparation dans le cas du sac à dos revient à partager deux arbres (poids et valeur) qui grandissent de couche en couche en fonction de si on prend l'objet ou non. L'ensemble des solutions sont stockés dans la dernière couche de chaque arbre.
Une séparation sera effectué à chaque objet. Donc dans notre cas on deux tableaux à doubles entrées poid et value. Dans poid[0] et value[0] on entre les résultats possible suite à cette séparation. Ceci est effectué par la fonction init. Dans poid[0][0] on entre le poids si on ajoute l'objet 1 et dans poid[0][1] le cas où on ne l'ajoute pas, de même pour value[0][0] et value[0][1].
La deuxième séparation va nécessiter que la deuxième couche soit 2 fois plus grande que la précédente car on va utiliser le même critère que pour la première séparation cependant à la suite des résultats possibles suite à celle ci.



On comprends donc très vite qu'avec un nombre important d'objets on peut atteindre des quantités de données qui dépassent ce qui est disponible en RAM. Donc il est nécessaire de garder en tête que le programme risque et va crash si le nombre d'objets est trop grand pour la machine à traiter.

3.2 Évaluation

Dans un problème d'optimisation, et dans notre cas qui est celui d'une maximisation d'un objectif, on a une borne qui correspond à une valeur qu'on souhaite atteindre ou du moins s'approcher au maximum de cette borne. Cette borne va exercée une contrainte sur notre dernière couche dans notre arbre des poids et va nous permettre d'évaluer parmi les cas obtenus, lesquelles sont réalisables.
On évalue donc chaque sommet et on retient leur position dans l'arbre car elle correspond à la position de leur valeur dans l'arbre value. En gardant en tête la restriction d'intégrité, Il y a plusieurs méthodes d'évaluations possible en fonction de comment on construit l'arbre avec le(s) critère(s) de séparation. Par exemple, il est possible de faire l'évaluation pendant la construction de l'arbre, cependant, cela nécessiterait des reallocs en C pour avoir une mémoire continue et éviter des lectures de valeurs erronées. Donc l'évaluation est effectuée à la fin de la construction de l'arbre, où on regarde chaque sommet si il est inférieur ou égale à la borne donnée en paramètre d'entrée.
Cette méthode d'évaluation nous permet d'avoir l'ensemble des solutions dans deux arbres entiers qui apportent chacun une information relative à un sommet, son poids et sa valeur.

3.3 Déterminer le chemin

Pour trouver le chemin qui a mené à un sommet en particulier, le problème dépend principalement du critère de séparation utilisé lors de la construction de l'arbre. Le critère de séparation utilisé est binaire, on prend l'objet ou non, ce qui nous permet d'assigner à chaque branchement un 0 ou un 1. Le 0 dans notre cas correspond à prendre l'objet et le 1 à ne pas le prendre. Et cela se fait sentir dans la position dans l'arbre dans le code aussi.
En effet, le code est écrit d tel sorte à ce que sur une couche, la position 0 va engendrée deux choix qui auront pour coordonnée 0 ou 1 dans la couche suivante. Tandis que engendra deux cas qui auront la position 2 ou 3 dans la couche suivante. Et en binaire ça donnerait 0 0 ou 0 1 dans le cas du 0 ou 1 0 et 1 1 dans le cas du 1. Et cela représente exactement les choix qu'on prends lors de la construction de l'arbre : 0 0 représente qu'on prend les 2 objets, 0 1 signifie qu'on prend le premier mais pas le deuxième, etc...

4 Conclusion

En informatique, le problème du sac à dos et ses dérivés sont encore beaucoup étudiés. Il en existe de nombreuses variantes, plusieurs fonctions objectif, etc... De nombreux algorithmes qui suivent ou non la contrainte d'intégrité sont proposés pour ce type de problèmes, comme la méthode des regrets pour le problème du voyageur.