Le problème du sac à dos

Présentation du problème (d'après http://interstices.info/jcms/c_19213/le-probleme-du-sac-a-dos)

L'énoncé de ce problème fameux est simple : *« Étant donné plusieurs objets possédant chacun un poids et une valeur et étant donné un poids maximum pour le sac, quels objets faut-il mettre dans le sac de manière à maximiser la valeur totale sans dépasser le poids maximal autorisé pour le sac ? »*

Toute formulation de ce problème commence par un énoncé des données. Dans notre cas, nous avons un sac à dos de poids maximal P et n objets. Pour chaque objet i, nous avons un poids p_i et une valeur v_i.

Pour quatre objets ($n = 4$) et un sac à dos d'un poids maximal de 30 kg ($P = 30$), nous avons par exemple les données suivantes :

Objets	1	2	3	4
v_i	7	4	3	3
p_i	13	12	8	10

Ensuite, il nous faut définir les variables qui représentent en quelque sorte les actions ou les décisions qui amèneront à trouver une solution. On définit la variable x_i associée à un objet i de la façon suivante : $x_i = 1$ si l'objet i est mis dans le sac, et $x_i = 0$ si l'objet i n'est pas mis dans le sac.

Dans notre exemple, une solution réalisable est de mettre tous les objets dans le sac à dos sauf le premier, nous avons donc : $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, et $x_4 = 1$.

Puis il faut définir les contraintes du problème. Ici, il n'y en a qu'une : la somme des poids de tous les objets dans le sac doit être inférieure ou égale au poids maximal du sac à dos.
Cela s'écrit ici $x_1.p_1 + x_2.p_2 + x_3.p_3 + x_4.p_4 \leq P$ et pour n objets :

$$\sum_{i=1}^{i=n} x_i.p_i \leq P$$

Pour vérifier que la contrainte est respectée dans notre exemple, il suffit de calculer cette somme : $0 \times 13 + 1 \times 12 + 1 \times 8 + 1 \times 10 = 30$, ce qui est bien inférieur ou égal à 30, donc la contrainte est respectée. Nous parlons alors de solution réalisable. Mais ce n'est pas nécessairement la meilleure solution.

Enfin, il faut exprimer la fonction qui traduit notre objectif : maximiser la valeur totale des objets dans le sac. Pour n objets, cela s'écrit :

$$\max \sum_{i=1}^{i=n} x_i.v_i$$

Dans notre exemple, la valeur totale contenue dans le sac est égale à 10. Cette solution n'est pas la meilleure, car il existe une autre solution de valeur plus grande que 10 : il faut prendre seulement les objets 1 et 2 qui donneront une valeur totale de 11. Il n'existe pas de meilleure solution que cette dernière, nous dirons alors que cette solution est optimale.

Méthodes de résolution
Il existe deux grandes catégories de méthodes de résolution de problèmes d'optimisation combinatoire : les méthodes exactes et les méthodes approchées. Les méthodes exactes permettent d'obtenir la solution optimale à chaque fois, mais le temps de calcul peut être long si le problème est compliqué à résoudre. Les méthodes approchées, encore appelées heuristiques, permettent d'obtenir rapidement une solution approchée, donc pas nécessairement optimale. Nous allons détailler un exemple d'algorithme de résolution de chaque catégorie.

Méthode approchée
Une méthode approchée a pour but de trouver une solution avec un bon compromis entre la qualité de la solution et le temps de calcul. Pour le problème du sac à dos, voici un exemple d'algorithme de ce type :

- calculer le rapport (v_i / p_i) pour chaque objet i ;
- trier tous les objets par ordre décroissant de cette valeur ;
- sélectionner les objets un à un dans l'ordre du tri et ajouter l'objet sélectionné dans le sac si le poids maximal reste respecté.

Déroulons cet algorithme sur notre exemple :

Première étape :

Objets	1	2	3	4
v_i / p_i	0,54	0,33	0,37	0,30

Deuxième étape :

Objets	1	3	2	4
v_i	7	3	4	3
p_i	13	8	12	10
v_i / p_i	0,54	0,37	0,33	0,30

Troisième étape :

Objet 1 : on le met dans le sac vide, le poids du sac est alors de 13 et inférieur à 30 ;
Objet 3 : on le met dans le sac car 13 + 8 = 21 est inférieur à 30 ;
Objet 2 : on ne le met pas dans le sac car le poids total (33) dépasserait 30.
Objet 4 : on ne le met pas dans le sac (poids total de 31).

La solution trouvée est donc de mettre les objets 1 et 3 dans le sac, ce qui donne une valeur de 10. Cette solution n'est pas optimale, puisqu'une solution avec une valeur totale de 11 existe. Néanmoins, cet algorithme reste rapide même si le nombre d'objets augmente considérablement.

Ce type d'algorithme est aussi appelé *algorithme glouton*, car il ne remet jamais en cause une décision prise auparavant. Ici, lorsque l'objet 2 ne peut pas entrer dans le sac, l'algorithme n'essaie pas d'enlever l'objet 3 du sac pour y mettre l'objet 2 à sa place.

Méthode exacte

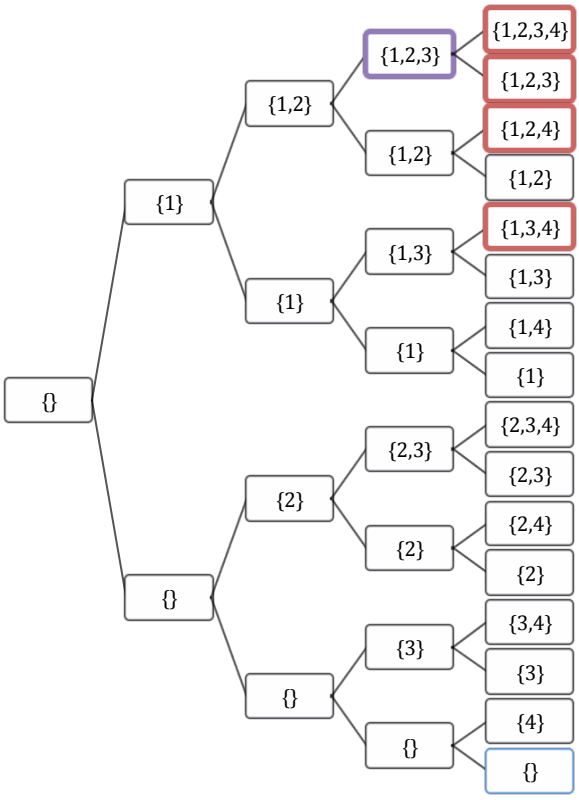
Pour trouver la solution optimale, et être certain qu'il n'y a pas mieux, il faut utiliser une méthode exacte, qui demande un temps de calcul beaucoup plus long (si le problème est difficile à résoudre). Il n'existe pas une méthode exacte universellement plus rapide que toutes les autres. Chaque problème possède des méthodes mieux adaptées que d'autres. Nous allons présenter un exemple d'algorithme de ce type, nommé *procédure par séparation et évaluation* (PSE), ou en anglais *branch and bound*. Nous n'exposerons ici qu'une version simplifiée d'une PSE.

Une PSE est un algorithme qui permet d'énumérer intelligemment toutes les solutions possibles. En pratique, seules les solutions potentiellement de bonne qualité seront énumérées, les solutions ne pouvant pas conduire à améliorer la solution courante ne sont pas explorées.

Pour représenter une PSE, nous utilisons un « *arbre de recherche* » constitué :

- de nœuds ou sommets, où un nœud représente une étape de construction de la solution
- d'arcs pour indiquer certains choix faits pour construire la solution.

Dans notre exemple, les nœuds représentent une étape pour laquelle des objets auront été mis dans le sac, d'autres auront été laissés en dehors du sac, et d'autres, encore, pour lesquels aucune décision n'aura encore été prise. Chaque arc indique l'action de mettre un objet dans le sac ou, au contraire, de ne pas le mettre dans le sac. La figure suivante représente l'arbre de recherche du problème donné en exemple.



On associe l'ensemble vide à la racine (dont la profondeur est 0) et pour un nœud de profondeur *i-1*, on construit deux fils : celui du haut où l'on ajoute l'objet *i* dans le sac et celui du bas où le sac reste tel quel. L'arbre de recherche achevé, chaque feuille représente alors une solution potentielle mais pas forcément réalisable. Dans le schéma, les feuilles au bord épais représentent les propositions irréalisables car supérieures au poids maximal à ne pas

dépasser. Pour déterminer la solution, il *suffit* de calculer la valeur du sac pour chaque nœud feuille acceptable et de prendre la solution ayant la plus grande valeur.

Cependant la taille de l'arbre de recherche est exponentielle en le nombre d'objets. Aussi il existe de nombreuses techniques algorithmiques de parcours de ce type d'arbre. Ces techniques ont pour but de diminuer la taille de l'arbre et d'augmenter la rapidité du calcul. Par exemple, on peut remarquer que le poids du nœud interne {1,2,3} dépasse déjà le poids maximal, il n'était donc pas nécessaire de développer l'étape suivante avec l'objet 4. Les procédures par séparation et évaluation (PSE) permettent d'élaguer encore plus cet arbre en utilisant des bornes inférieures et supérieures de la fonction objectif :

- Une borne inférieure est une valeur minimum de la fonction objectif. Autrement dit, c'est une valeur qui est nécessairement inférieure à la valeur de la meilleure solution possible. Dans notre cas, une configuration du sac réalisable quelconque fournit une borne inférieure.
- Une borne supérieure est une valeur maximale de la fonction objectif. Autrement dit, nous savons que la meilleure solution a nécessairement une valeur plus petite. Dans notre cas, nous savons que la valeur de la meilleure solution est nécessairement inférieure à la somme des valeurs de tous les objets (comme si on pouvait tous les mettre dans le sac).

Supposons maintenant que la borne inférieure soit initialisée par l'algorithme heuristique vu précédemment. Pendant la recherche à chaque nœud, nous pouvons déterminer une borne supérieure : la somme de toutes les valeurs de tous les objets déjà mis dans le sac plus la somme des valeurs des objets restants dont on ne sait pas encore s'ils seront dans le sac. À partir d'un nœud et de sa borne supérieure, nous savons que les solutions descendantes de ce nœud ne pourront pas avoir une valeur plus grande que cette borne supérieure. Si jamais, à un nœud donné, la valeur de la borne supérieure est inférieure à la valeur de la borne inférieure, alors il est inutile d'explorer les nœuds descendants de celui-ci. On dit qu'on *coupe* l'arbre de recherche. En effet, si à partir d'un nœud, nous savons que nous ne pourrons pas faire plus de 10 (borne supérieure calculée) et que la borne inférieure existante est à 11 (on a déjà une solution de valeur 11), alors les solutions descendantes de ce nœud ne sont pas intéressantes. Enfin, dernière remarque, la valeur de la borne inférieure peut être actualisée lorsqu'est trouvée une solution réalisable qui possède une valeur plus grande.

Ce système de calcul de bornes demande un faible coût de temps de calcul, et permet d'augmenter la rapidité de la PSE puisqu'elle coupe des branches d'arbre pour ne pas perdre de temps à les explorer. D'autres techniques servent à diminuer la taille de l'arbre et à augmenter la rapidité. Par exemple, elles sont basées sur l'ordre dans lequel on prend les décisions sur les objets, ou sur une évaluation à chaque nœud, ou encore sur des propriétés du problème qui permettent de déduire des conclusions sur certains objets. Sur notre exemple initial, il est possible d'obtenir de façon exacte la solution optimale en n'examinant que 9 nœuds au lieu de 31.

Travail demandé

Il s'agit tout d'abord de comparer différentes méthodes pour résoudre des instances du problème du sac à dos selon quatre méthodes :

- méthode approchée gloutonne ;
- méthode énumérative (décrite ci-après) ;
- méthode récursive (sans optimisation);
- méthode PSE.

Le premier objectif est de présenter lors du premier TD une implémentation de la méthode gloutonne. Dans un premier temps vous pourrez vous concentrer sur la mise au point des algorithmes : l'énoncé des données pourra être contenu dans un tableau constant et le code source pourra être concentré dans un seul fichier. Ensuite, dans la version finale, vous proposerez une organisation modulaire des sources afin d'améliorer votre code et la qualité de votre travail en terme de lisibilité, maintenabilité, portabilité, extensibilité et réutilisabilité. Il s'agira aussi de proposer un programme de démonstration dont le synopsis est le suivant :

resoudre-sac-a-dos chemin poids-maximal methode
où

- chemin : le chemin du fichier texte contenant l'énoncé du sac à dos à résoudre, chaque ligne contiendra le nom d'un objet (une séquence d'au plus 30 caractères sans espace au sens large), son poids et sa valeur (deux nombres réels) ;
- poids_maximal : un nombre réel ;
- methode : gloutonne, enumerative, recursive, pse.

La méthode itérative se base sur une bijection entre les parties d'un ensemble de n éléments et les entiers naturels inférieurs à 2^n : il suffit d'identifier la présence du ième objet au ième bit de l'écriture binaire. Ainsi le vecteur {true,true,true,false} de l'exemple précédent correspond à l'entier 1110_2 c'est à dire 14_{10}. Ainsi pour énumérer (passer en revue) tous les sous ensembles, on peut utiliser un indice de boucle parcourant les entiers compris entre 0 (correspondant à l'ensemble vide) et 2^{n-1} (l'ensemble initial). Mais plutôt que traduire à chaque tour de boucle l'entier en ensemble, on peut directement réaliser l'opération d'incrémentation sur le vecteur de booléens. Par exemple, pour passer de 1 à 4 (soit 0001 → 0010 → 0011 → 0100) :

{false, false, false, true} → {false, false, true, false } → {false, false, true, true} → {false, true, false, false}

Organisation

Le projet est travaillé et étudié en trinôme mais la notation est individuelle. L'enseignant évaluera la contribution de chacun des éléments du trinôme au travail commun. L'enseignant se réservera la possibilité de modifier la composition de chacun des trinômes. Afin de permettre un travail profitable, il est conseillé de ne pas créer de groupes avec des niveaux trop différents.

TD - Chaque semaine, deux jours avant le TD, une brève synthèse (1/2 page environ) des travaux effectués depuis la séance précédente sera transmis (mail ou papier) à l'enseignant. On précisera les points que l'on désire plus particulièrement aborder avec l'enseignant chargé de suivre le projet. Ce courrier comprendra aussi le code source. Tous les documents numériques seront regroupés dans un unique fichier au format pdf (voire postscript). Chaque document contiendra la date et les noms du trinôme.

Soutenance - La présentation finale du projet se fera en salle de TD. Cette présentation se fera autour d'une démonstration et d'un compte rendu écrit présentant :

- les objectifs atteints et ceux qui ne le sont pas ;
- l'exposé d'un problème technique rencontré et sa résolution ;
- quelques exemples montrant que vous avez su éviter des redondances dans votre code.

Il s'agira aussi de bien préciser l'origine de toute portion de code empruntée (sur internet, par exemple) ou réalisée en collaboration avec tout autre trinôme. Il est évident que tout manque de sincérité sera lourdement sanctionné.

Notation - La note est composée de 3 éléments. Ces trois éléments sont cumulatifs (pour obtenir la note finale, on ajoute les différents éléments)

- Contrôle continu (40/200) : une note sur 8 est attribuée à l'issue de chaque séance de TP. Cette note sera établie à partir des critères suivants: respect des consignes ; évaluation de la synthèse et du code fourni ; travail accompli depuis la séance précédente ; évolution du projet.

- Soutenance (80/200) : une note sur 80 est attribuée à l'issue de la soutenance. Cette note sera établie à partir des critères suivants:
  - Évaluation de la présentation orale et la démonstration de l'exécution (10 pts)
  - Réponses aux questions posées sur le projet (10 pts)
  - Évaluation du rapport sur le projet (20 pts)
  - Qualité du code fourni (40 pts) (non duplication de code, concision et lisibilité du code, modularité et qualité du découpage, généricité, réponse aux spécifications, extensions éventuelles, présence d'un fichier *makefile*)

- Devoir surveillé (80/200)

Présence - Les présences aux TD, à la soutenance et au devoir surveillé sont obligatoires. Toute absence injustifiée donnera la note 0 pour l'épreuve concernée. En cas de circonstances exceptionnelles, contacter son enseignant de TD par mail au plus tard le jour de la séance de TD concernée ou le jour du devoir surveillé.

Groupe	Enseignant	Adresse électronique	TD
CSB4A1	S. Gueorguieva	stefka@labri.fr	mercredi 14h
CSB4A2	M.C. Counilh	counilh@labri.fr	mardi 14h
CSB4A3	B. Mery	mery@labri.fr	mardi 14h
CSB4A4	P.A. Wacrenier	wacrenier@labri.fr	vendredi 9h30
MHT63	F. Broquedis	broquedis@labri.fr	vendredi 14h

Université Bordeaux 1 Licence
INF 255
Examen seconde session juin 2008 Durée : 1h30 – Sans document

Projet de programmation 2

On utilisera le langage C pour répondre aux questions.

Implémentation des exceptions en C

1. Rappeler le prototype et le rôle des deux fonctions sur lesquelles se base le mécanisme d'exception en C vu en cours.
2. Quelle structure de données est nécessaire à toute implémentation du mécanisme d'exception?

Le problème du sac à dos. *Les questions 3 – 4 - 5 sont indépendantes.*
On dispose d'un sac capable de supporter un poids (en gramme) maximal et d'un ensemble d'objets ayant chacun un poids et une valeur (en euro). L'objectif est de remplir le sac d'objets sans dépasser le poids maximal tout en maximisant la somme des valeurs des objets qu'il contient.
Exemple : on dispose d'un sac supportant 3 kg et des objets suivants :

Bague	12g	3400€
Tournevis	270g	7€
Ordinateur	2500g	1234€
Baguette	250g	0,95€

Ne pouvant emporter tous les objets, on sélectionnera les objets « Bague », « Tournevis » et « Ordinateur » pour un montant de 4641€ et un poids de 2782g.

1. Définir un type *objet* afin de mémoriser le nom, le poids et la valeur d'un objet.

2. Définir un type *sac_a_dos* capable de mémoriser l'ensemble initial d'objets (un vecteur d'objets), le poids maximal du sac à dos et de décrire la sélection retenue (sous ensemble des objets, poids et montant du sac).

NB. Pour mémoriser l'ensemble des objets retenus on pourra associer un vecteur de booléens au vecteur d'objets. Ainsi, la solution de l'exemple précédent peut être représentée par le vecteur {true,true,true,false} en référence au tableau d'objets ci-dessus.

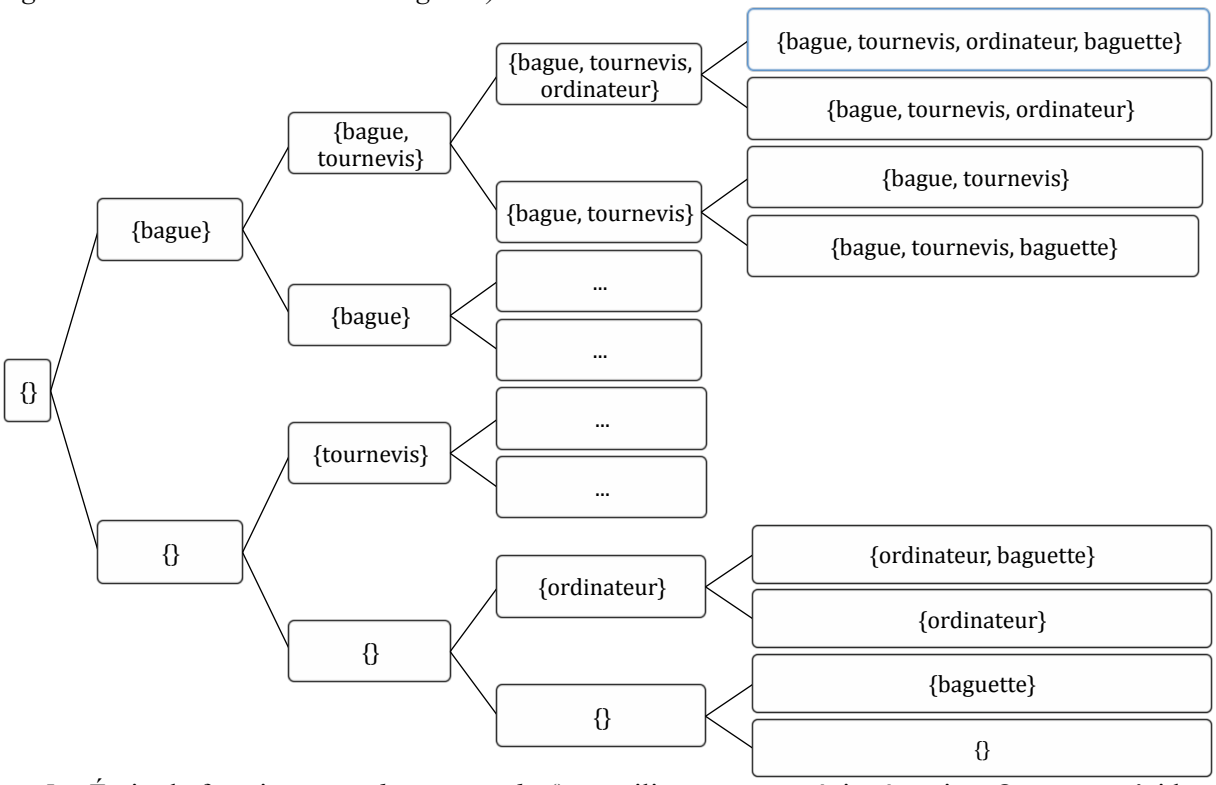
Il s'agit de résoudre le problème du sac à dos en testant systématiquement tous les sous ensembles de l'ensemble de départ. Dans ce devoir nous allons programmer deux techniques : une itérative, l'autre récursive.
La méthode itérative se base sur une bijection entre les parties d'un ensemble de n éléments et les entiers naturels inférieurs à 2^n : il suffit d'identifier la présence du $i^{ème}$ objet au $i^{ème}$ bit de l'écriture binaire. Ainsi le vecteur {true,true,true,false} de l'exemple précédent correspond à l'entier 1110_2 c'est à dire 14_{10}. Ainsi pour énumérer (passer en revue) tous les sous ensembles, on peut utiliser un indice de boucle parcourant les entiers compris entre 0 (correspondant à l'ensemble vide) et 2^n-1 (l'ensemble initial). Mais plutôt que traduire à chaque tour de boucle l'entier en ensemble, on peut directement réaliser l'opération d'incrémentation sur le vecteur de booléens. Par exemple, pour passer de 1 à 4 (soit 0001 → 0010 → 0011 → 0100) :
{false, false, false, true}→{false, false, true, false }→{false, false, true, true}→{false, true, false, false}

3. Écrire la fonction bool incrementer(bool *bit, int nombre_de_bits) qui transforme le tableau bit en utilisant l'algorithme d'incrémentation binaire et retourne true si et seulement si la configuration obtenue contient au moins une fois la valeur true.

4. En utilisant la fonction incrémenter, écrire la fonction résoudre_sac_a_dos() qui prend en paramètre un sac à dos et qui calcule une (la) meilleure solution possible.

La méthode récursive utilise un schéma de résolution récursif analogue à celui employé en cours pour résoudre le problème des n-reines. Ici, il s'agit d'explorer (sans le construire) un arbre

binaire complet où à chaque niveau on choisit de sélectionner ou non un objet (ici suivant l'ordre bague – tournevis – ordinateur – baguette) :



5. Écrire la fonction *resoudre_sac_a_dos*() en utilisant une stratégie récursive. On pourra s'aider en définissant des fonctions auxiliaires.

On souhaite accélérer la vitesse de traitement en supprimant le plus possible les calculs inutiles. Par exemple, pour le voyageur de commerce on peut arrêter une exploration dès lors que la longueur de la tournée en cours d'exploration est plus longue que la plus petite tournée calculée jusqu'à lors.

6. Proposer une optimisation basée sur la valeur des objets restant à prendre. Montrer comment cette optimisation peut s'intégrer à la version récursive. Peut-on optimiser de la sorte la version itérative ?