Le problème du *bin packing*

Présentation du problème (inspiré de http://interstices.info/sac-a-dos)

On considère un ensemble de sacs identiques et un ensemble d'objets dont on connaît le poids. Sachant que les sacs ne peuvent supporter qu'un poids maximum, combien faudra-t-il au minimum de sacs pour y ranger l'ensemble des objets considérés ? Ce problème d'optimisation, connu sous le nom de « *bin packing problem* » - problème du conditionnement dans des boîtes - a suscité des milliers d'articles de recherche et apparaît aux caisses des supermarché, lorsque l'on veut sauvegarder sans les couper un ensemble de fichiers (des divx) sur des supports identiques (des dvd) ou encore lorsqu'il s'agit de répartir un ensemble de flux sur différents canaux (fibres optiques, porteuses hertziennes).

Toute formulation de ce problème commence par un énoncé des données. Dans notre cas, nous avons n objets et un nombre non borné (c'est à dire suffisant) de sacs supportant une charge maximale P. Pour chaque objet i, nous avons un poids p_i.

Pour quatre objets ($n = 7$) et des sacs de capacité maximale de 10 kg ($P = 10$), nous avons par exemple les données suivantes :

Objets	1	2	3	4	5	6	7
p_i	3	4	4	3	3	2	1

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 S correspondant au nombre de sac utilisés et la variable x_i associée à un objet i de la façon suivante : $x_i = k$ si l'objet i est mis dans le sac numéro k.

Dans notre exemple, une solution réalisable avec S=3 est de mettre les 2 premiers objets dans un premier sac, les trois suivants dans le deuxième sac et les deux derniers dans un troisième :

Objets	1	2	3	4	5	6	7
p_i	3	4	4	3	3	2	1
x_i	1	1	2	2	2	3	3

Puis il faut définir les contraintes du problème. Ici, il n'y en a qu'une : pour chaque sac la somme des poids des objets qu'il contient doit être inférieur ou égal au poids maximum. Pour vérifier que la contrainte est respectée dans notre exemple, il suffit de vérifier que le poids de chaque sac est bien inférieur ou égal à 10, ici on a 3 + 4 , 4 + 3 + 3 et 2 + 1 donc la contrainte est respectée. Nous parlons alors de solution *réalisable*. Mais ce n'est pas nécessairement la meilleure solution.

Enfin, il faut préciser la fonction qui traduit notre objectif : minimiser le nombre total de sacs. La solution apportée à l'énoncé précédant n'est pas la meilleure, car il existe une solution à deux sacs : {1,2,4} et {3,5,6,7}. Il n'existe pas de meilleure solution que cette dernière puisqu'il faut au moins 2 sacs pour supporter les 20 kg : cette solution est dite *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.

Méthodes approchées

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 *bin packing*, on peut utiliser l'algorithme naïf qui consiste à remplir les sacs les uns après les autres en prenant les objets les uns après les autres. C'est cet algorithme qui a été employé pour traiter l'exemple introductif. Le Best Fit Deacressing – BFD – (meilleur remplissage par ordre décroissant) améliore l'algorithme naïf en :

- triant tous les objets par ordre décroissant de poids ;
- sélectionnant les objets un à un dans l'ordre du tri et en ajoutant l'objet sélectionné dans le sac le plus lourd tel que la contrainte de poids maximal reste respectée.

Déroulons cet algorithme sur notre exemple :

Première étape :

Objets	2	3	1	4	5	6	7
p_i	4	4	3	3	3	2	1

Deuxième étape :

Objets	2	3	1	4	5	6	7
p_i	4	4	3	3	3	2	1
x_i	1	1	2	2	2	1	2

Cette solution est optimale, mais cet algorithme est imparfait comme le montre le cas suivant :

Objets	2	3	1	4	5	6
p_i	4	4	3	3	3	3
x_i	1	1	2	2	2	3

La qualité de cet algorithme est de rester rapide même si le nombre d'objets augmente considérablement tout en approchant de moins d'un facteur 2 la solution optimale (pourquoi ?). Aussi de nombreux algorithmes approchés ont été mis au point, étudiés et comparés parmi ceux-ci citons :

- First Fit - FF : on place au fur et à mesure les objets dans le premier sac possible ;
- Worst Fit - WF : on place au fur et à mesure les objets dans le sac le plus léger possible ;
- Best Fit – BF : on place, au fur et à mesure, les objets dans le sac le plus lourd possible ;
- First Fit Decreasing et Worst Fit Decreasing – FFD et WFD : on trie les objets dans l'ordre des poids décroissant puis on applique FF / WF;

Ce type d'algorithme est aussi appelé *algorithme glouton*, car il *mange* les données au fur et à mesure et ne remet jamais en cause une décision prise auparavant. Dans le dernier exemple, lorsque l'objet 1 ne peut pas entrer dans le premier sac, l'algorithme n'essaie pas d'enlever l'objet 3 du sac pour y mettre l'objet 1 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*

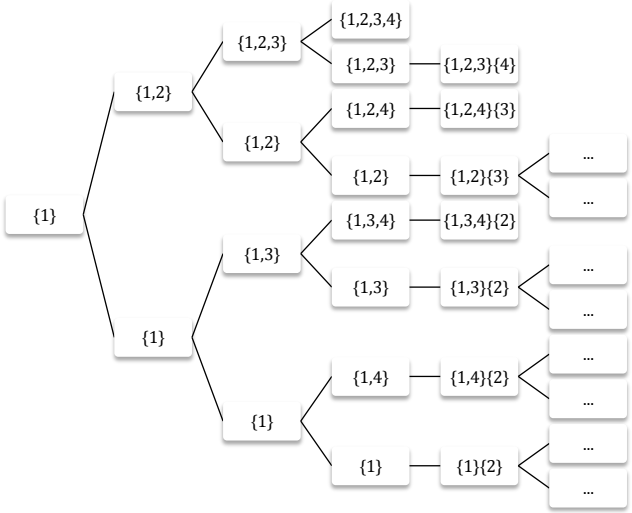
(PSE), ou en anglais *branch and bound*. Nous n'exposerons ici qu'une version simplifiée d'une PSE.

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 les sacs et d'autres pour lesquels aucune décision n'aura encore été prise. Chaque arc indique l'action de mettre un objet dans le sac courant ou, au contraire, de ne pas le mettre dans le sac. Les nœuds sont étiquetés par une liste d'ensemble chacun correspondant à un sac, de plus le dernier ensemble de la liste correspond au sac que l'on est en train de remplir. La figure suivante représente l'arbre de recherche du problème donné en exemple.



Au départ, on cherche à remplir le premier sac. Tout d'abord on lui affecte l'objet 1 : comme il faudra bien que cet objet soit contenu dans un sac, on décide en quelque sorte de nommer 1 le sac le contenant. La racine de l'arbre, dont la profondeur est par définition 0, contiendra donc l'ensemble {1} qui représente le contenu du 1er sac. Puis pour un nœud de profondeur i on construit deux fils : celui du haut où l'on ajoute l'objet i+1 dans le sac et celui du bas où le sac reste tel quel. Lorsque l'on a considéré tous les objets pour un sac donné, on poursuit la construction de l'arbre en passant au sac suivant et en lui affectant initialement l'objet non sélectionné dont l'indice est le plus petit. On poursuit ainsi la construction de l'arbre jusqu'à avoir rangé tous les objets. Dans l'arbre de recherche achevé, chaque feuille représente 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 le nombre de sacs pour chaque nœud feuille acceptable et de prendre la solution ayant la plus petite valeur.

Cependant la taille de l'arbre de recherche est exponentielle en le nombre d'objets, et *il n'est pas question d'implanter un tel arbre en mémoire*. Aussi il existe de nombreuses techniques algorithmiques de parcours de ce type d'arbre. Ces techniques ont pour but et d'augmenter la rapidité du calcul en diminuant la taille de l'arbre de recherche. 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. Les 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, ce peut être la somme des poids des objets divisés par la capacité des sacs.
- 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 au nombre d'objets (comme si on les mettait chacun dans un sac) ou mieux la meilleure solution doit être strictement inférieure à la valeur trouvée par l'algorithme BFD car nous cherchons à améliorer le meilleur résultat connu.

Supposons maintenant que la borne supérieure soit initialisée par l'algorithme BFD. Pendant la recherche à chaque nœud, nous pouvons déterminer une borne inférieure : le nombre de sacs déjà remplis plus la somme de tous les poids de tous les objets restant divisée par le poids maximal. À partir d'un nœud et de sa borne inférieure, nous savons que les solutions descendantes de ce nœud ne pourront pas avoir une valeur plus petite que cette borne. Si jamais, à un nœud donné, la valeur de la borne inférieure est supérieure ou même égale à la valeur de la borne supérieure, 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 moins de 5 sacs (borne inférieure calculée) et que la borne supérieure existante est à 4 (on a déjà une solution de valeur 4), alors les solutions descendantes de ce nœud ne sont pas intéressantes. Enfin, dernière remarque, la valeur de la borne supérieure peut être actualisée lorsqu'est trouvée une solution réalisable qui possède une valeur plus petite.

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 peuvent être utilisées à diminuer la taille de l'arbre en jouant par exemple 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. Dans notre cas, il est par exemple inutile de prendre un nouveau sac alors qu'on aurait pu continuer à remplir le sac courant.

Travail demandé

Il s'agit tout d'abord de comparer différentes méthodes pour résoudre des instances du problème du *bin packing* selon trois classes de méthodes :

- méthode approchée gloutonne (FF, BF, WF, FFD, BFD, WFD) ;
- méthode récursive (sans optimisation);
- méthode PSE.

Le premier objectif est de présenter lors du TD de la semaine 4 une implémentation de quelques méthodes gloutonnes. 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-bin-packing chemin poids-maximal methode

où

- chemin : le chemin du fichier texte contenant l'énoncé du bin packing à résoudre, chaque ligne contiendra le nom d'un objet (une séquence d'au plus 30 caractères sans espace au sens large), son poids (un nombre réel) ;
- poids_maximal : un nombre réel ;
- methode : FF, BF, WF, FFD, BFD, WFD, recursive, pse.

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 (grâce aux commandes a2ps et ps2pdf). 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 et rapport (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
  - Réponses aux questions posées sur le projet
  - Évaluation du rapport sur le projet
  - Qualité du code (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) : ce devoir portera sur des questions liées aux deux projets, il sera nécessaire de maitriser parfaitement le code des projets pour réussir cet examen.

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. Notons qu'en cas d'absence justifiée au devoir surveillé l'étudiant se verra proposer une épreuve de substitution.

Groupe	Enseignant	Adresse électronique	TD
CSB4A1	S. Gueorguieva	stefka@labri.fr	mercredi 14h
CSB4A2	M.C. Counilh	counilh@labri.fr	jeudi 14h
CSB4A3	N. Bonichon	bonichon@labri.fr	mardi 14h
CSB4A4	P.A. Wacrenier	wacrenier@labri.fr	vendredi 14h
MHT63	N. Bonichon	bonichon@labri.fr	vendredi 14h