Université de Lille 1 SCIENCES ET TECHNOLOGIES

UFR IEEA

AAC

2014–2015
Master1 d'Informatique

TP - Heuristiques gloutonnes

Objectifs:

. Abstraire la notion d'heuristique gloutonne et l'implémenter pour plusieurs problèmes

. Implémenter plusieurs heuristiques gloutonnnes pour le même problème et les expérimenter.

On va continuer à travailler sur les problèmes vus dans le TP précédent $BINPACKING$, $Partition$, Sum mais en considérant les problèmes d'optimisation définis comme suit:

$BINPACKING$ Donnée:
n –un nb d'objets
$x_1, \cdots, x_n$ – n entiers, les poids des objets
c –la capacité d'un sac (entière)
Sortie:
$aff : [1..n] \rightarrow [1..n]$ tq:
$\sum_{i/aff(i)=j} x_i \leq c$, pour tout numéro de sac j, $1 \leq j \leq k$.
Fonction Objectif à minimiser: le nombre de sacs utilisés i.e. $sup_{i=1}^{n} aff(i)$.
SUM
Donnée:
n –un nb d'objets
$x_1, \cdots, x_n$ – n entiers, les entiers
s –la cible (entière)
Sortie:
$aff : [1..n] \rightarrow [0..1]$ tq:
$\sum_{i/aff(i)=0} x_i \leq s$
Fonction Objectif à maximiser: $\sum_{i/aff(i)=0} x_i$
$Partition$ Donnée:
n –un nb d'objets
$x_1, \cdots, x_n$ – n entiers, les entiers
Sortie:
$aff : [1..n] \rightarrow [0..1]$
Fonction Objectif à minimiser: $max(\sum_{i/aff(i)=0} x_i, \sum_{i/aff(i)=1} x_i)$

Les problèmes de décision associés sont NP-complets (cf précédents TPS).

On va donc utiliser des heuristiques pour construire "rapidement" des solutions peut-être non optimales. On va utiliser 3 heuristiques:

Heuristique 1 (NextFit) Le principe est simple; on a un sac courant; à chaque objet, on le met dans le sac courant si il y a assez de place. Sinon , on crée un nouveau sac courant et on y met l'objet;(cf cours)

Heuristique 2 (FirstFit) Pour chaque objet, on regarde si il rentre dans un des sacs créés: si oui, on le met dans le premier qui convient; sinon, on créee un nouveau sac et on y met l'objet;

Heuristique 3 (BestFit) Pour chaque objet, on regarde si il rentre dans un des sacs créés: si oui, on le met dans le celui qui est le plus rempli parmi ceux qui conviennent. Sinon, on crée un nouveau sac et on y met l'objet.

De plus pour chacune de ces heuristiques, on peut les appliquer soit en "online"- on prend les objets dans l'ordre d'énumération donné en entrée-, soit en "offline" et en triant les objets par poids décroissants.

A Faire: (rendu sous Prof d'une archive avec le code et le compte-rendu)

.Implémenter ces 3 heuristiques

.Implémenter ces heuristiques en triant les objets par poids décroissants

.Les expérimenter et les comparer sur quelques exemples à la fois quant à la qualité des solutions retournées et le temps d'exécution.

.Implémenter une heuristique de votre choix sur SUM ou $Partition$.

Voici une proposition d'architecture "assez abstraite" en JAVA dont vous pouvez vous inspirer ... ou pas.

```
/*********L'objet problème***************/
public class Pbl {
public Pbl(){} }
//le pb d'optimisation du BinPacking
public class PblBinPack extends Pbl{
    public int nbobjets; //nb d'objets
    public int poids[];  //poids des objets
    public int cap;       //capacité d'un sac      ...    }
}
/*********L'objet solution partielle***************/
/* Interface des solutions partielles */
interface PartialSolution {
    public boolean Complete();  //solution est complete!
    public void Display();      //affichage}
/* classe des solutions partielles pour BinPacking */
...class PartialSolutionBinPack implements PartialSolution{...
/* attention: pour représenter une solution partielle  privilégiez la simplicité!
   doit contenir des méthodes pour prolonger une solution: affecter un sac à un objet, ...)*/ }
/*********L'objet "constructeur  de solutions partielles"
   construit une solution de facon incrémentale
   peut etre vu comme un "walker" dans un arbre de solutions partielles
   à la création il est à la racine;
   à chaque pas, passe à un noeud voisin selon un critère dépendant de l'heuristique choisie
   la balade est terminée quand la solution est complète ***************/
/* interface du parcoureur de solutions */
interface SolutionWalker {
    public boolean Terminated();  //la solution courante est complete
    public void NextPartialSolution(); //passe a UNE solution suivante
    public PartialSolution Current(); //la solution courante}

/* classe abstraite du parcoureur de solution pour BinPacking */
....abstract class SolutionWalkerBinPack implements SolutionWalker {  ...     }

/* classe  du parcoureur de solution pour bin_packing selon un critère glouton XX
et avec choix de l'énumération */
...class SolutionWalkerBinPackXX extends SolutionWalkerBinPack{

public SolutionWalkerBinPackXX(Pbl_Bin_Pack pbl, EnumerationObjets en){
//on peut passer en en parametre: deux énumérations ici:
// online et par poids décroissants
//on peut aussi simplement construire le probleme avec objets tries au depart.... }

public void NextPartialSolution(){ //implémenter selon le critère glouton!} }
/******************/
/* interface d'une enumeration d'objets -on peut utiliser aussi l'interface Java Iterator */
interface EnumerationObjets{
     boolean hasMoreElements ();
     int NextElement (); }
/* classe d'enumeration online */
..class EnumerationOnline implements Enumeration_objets{   //... }
/* classe d'enumeration triee selon les poids décroissants */
 ...class EnumerationTriee implements EnumerationObjets{
 // on utilisera par exemple une implémentation de la classe Comparator de java.util }
```