PROJET RECHERCHE OPERATIONNEL

Problème de Sac à Dos

ELABORE PAR :

Marwen KRAIEM

2017-2018

Problème du Sac à Dos

Le problème du Sac à Dos représente une situation dans laquelle une personne dispose de n objets (avec un poids et une utilité) et d'un sac ayant une capacité maximale. Le problème est de trouver un ensemble d'objets dont la somme des utilités est maximale sans dépasser la capacité du sac.

Le problème de sac à dos multidimensionnel ((0-1) MKP) est un problème NP-complet, permettant de modéliser de nombreux problème de chargement optimal. Il consiste à maximiser une fonction linéaire de variables bivalentes (0,1) soumises à des contraintes linéaires d'inégalités dont tous les coefficients sont entiers et positifs :

$$(\mathrm{P1})\begin{cases} max\ cx \\ sc \\ Ax \leqslant b \\ x \in \{0,1\}^N \end{cases}$$

A : est une matrice dense de contraintes, de dimension $M \times N$ à coefficients entiers positifs.

b : le vecteur des seconds membres de contraintes, de dimension M, à coefficients entiers positifs.

c : un vecteur fonction économique de dimension N à coefficients entiers positifs.

Afin de bien appréhender la dynamique des AG, introduisons les principaux éléments utilisés dans ce papier.

- Individu ou chromosome : toute solution potentielle $\underline{X} = (x_1,x_2,...,x_N)$,
- Population : un ensemble d'individus de taille $Taille\ Pop$
- Fitness : la valeur sélective de l'individu :

$$f_k = \sum_{i=1}^{N} C_i x_i \ \text{ avec } k = 1...Taille\_Pop$$

- Solution réalisable : tout individu $\underline{X}$ qui vérifie :

$$\sum_{j=1}^{N} a_{ij} x_j \leqslant b_i$$

avec $i = 1...M$

$x_j \in \{0,1\}$ avec $j = 1...N$

N : nombre de variables

M : nombre de contraintes.

Application de l'AG au problème du sac à dos

Dans un premier temps il faut poser la terminologie de l'algorithme génétique que nous allons utiliser pour approcher une solution pour le problème du sac à dos.Les algorithmes génétiques se basant sur populations (ou liste d'individus), on obtient les terminologies suivantes :

- Population : liste de sacs.
- Individu : sac contenant des objets.
- Individu valide : sac dont la somme des poids des objets ne dépasse pas le poids maximum du sac.
- Individu non valide : sac dont la somme des poids des objets dépasse le poids maximum du sac.
- Chromosome : somme des valeurs des objets du sac.
- Fitness : obtention du meilleur chromosome possible, donc maximisation de la somme des valeurs des objets du sac en ne dépassant pas le poids maximum.

On va donc travailler sur une liste de sac. Il faut garder è l'esprit que l'on va approcher un résultat au fur et è mesure. Une grande part d'aléatoire étant è prendre en compte, on peut l'atteindre rapidement ou au bout de 1000 générations.

1. Description de l'algorithme génétique

Etant donné que les algorithmes génétiques laissent une grand part è la customisation du modèle utilisé, il faut d'abord fixer les paramètres d'utilisation du modèle. On dispose d'un grand nombre d'outils pour la modification ou la sélection d'individus. Plusieurs modèles ont été essayés plus ou moins complexe pour que ce soit pour l'évolution d'une population ou les caractéristiques de la population de base.

L'évaluation se fait donc sur le calcul présent sur la première ligne. Il met en pratique le rapport suivant :

- l'objet se verra attribué une bonne évaluation s'il a une valeur importante et un poids peu éloigné du poids maxi.
- l'objet se verra attribué une mauvaise évaluation s'il a une valeur importante et un poids éloigné du poids maximum.
- Si l'objet a une valeur faible, quel que soit son poids, il est peu probable que son évaluation soit bonne.

Ensuite, il faut fixer les méthodes de sélection et les portions de population qui seront affectées aux croisements, aux mutations, et les autres qui seront gardées pour les générations suivantes.

Une fois l'évaluation terminée et les sacs triés (par ordre croissant), les 30 pourcents premiers sont gardés tels quels et les autres passeront dans un algorithme de sélection.

L'opérateur de sélection qui a été choisi est le tournoi. Les combats de ce tournoi s'effectueront entre 6 sacs. Ils sont choisis au hasard parmi les sacs restants pour avoir à chaque fois 6 sacs d'évaluation quelconque. Cela permet à des sacs qui ne sont pas forcément bon au départ, d'avancer dans les générations, avec un triage aléatoire de 6 sacs faibles. Seuls 2 des 6 sacs seront déclarés vainqueurs et seront placés dans les élus pour la génération suivante.

Quand tous les combats sont terminés, 30 pourcents des perdants passeront dans un opérateur de croisement. Cet opérateur fonctionne comme ceci :

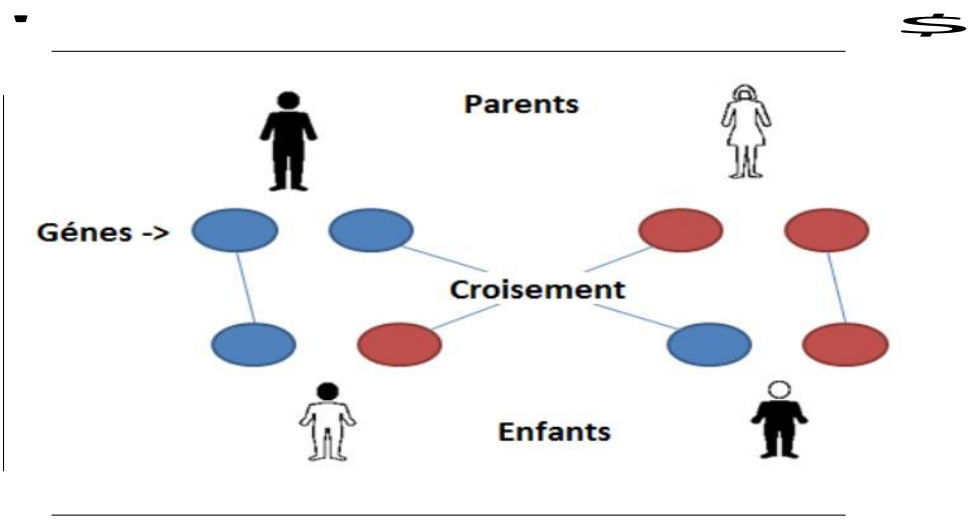

Figure 6 – Principe de croisement --

A partir de deux sacs pères, on crée deux sacs fils qui seront composés de moreaux des pères. Et enfin, pour tous les sacs restants, on utilise l'opérateur de mutation. Cet opérateur a deux actions possibles suivant les cas qui s'offrent à lui :

- Si le poids courant est inférieur au poids maximum, alors deux objets créés aléatoirement sont ajoutés dans le sac.
- Si le poids courant est supérieur au poids maximum, alors deux objets créés aléatoirement sont supprimés du sac.

2. Explication de la structure de l'algorithme

i. Fonction principale

L'entête de la fonction principales est :
[solutionOptimale,profitSolutionOptimale,poidsSolutionOptimale,tempsExecution,affichObjetSol] = Main2(k,a,c,d,e,f,g)
Avec

- k= 1..10 : le numéro de l'instance
- a= coeffTaillePopulationTailleChromosome : ce coefficient permet de définir la taille de la population.En effet :
- taille de population= taille de chromosome* coeffTaillePopulationTailleChromosome
- c= coefSelection : c'est la proportion de la population à sélecter.
- d= coeffCroisement : c'est la proportion de la population qui va subir un croisement. – e= coeffMutation : c'est la proportion de la population qui va subir un croisement.
- f= nbIterations : Ce paramètre représente le nombre d'itérations de l'algorithme qui constitue le critère d'arrêt d'une exécution de l'algorithme.
- g= nbRepetition : C'est le nombre de répétition de l'exécution de l'algorithme. A la fin de l'algorithme, on choisit la meilleure solution parmi les g solutions trouvées.

```
function[solutionOptimale,profitSolutionOptimale,poidsSolutionOptimale,tempsExecution
,affichObjetSol ] = Main2(k,a,c,d,e,f,g )
kp=k;
if(kp==1)
    profit=[5 8 6 3 7 5 2 9 2 7];
    poids=[2 1 1 8 9 3 4 3 6 3];
    poidsSac=27;
end
if(kp==2)
profit=[5 6 7 8 6 4 9 7 8 4];
poids=[10 7 9 8 1 1 7 7 7 3];
poidsSac=20
end
if(kp==3)
profit=[2 1 2 6 10 4 3 10 2 9 8 2 6 6 5 9 8 10 5 8];
poids=[2 7 6 3 1 9 8 8 8 3 5 2 6 3 5 4 9 4 9 3];
poidsSac=58; end
if(kp==4) profit=[1 9 6 4 5 9 7 7 10 1 4 7 8 1 8 10 3 4 2 10];
poids=[8 8 6 10 10 3 1 7 3 6 1 1 6 7 1 8 3 10 6 4];
poidsSac=38; end
if(kp==5)
profit=[4 8 3 9 8 5 3 7 7 8 6 7 7 5 10 2 3 5 8 5 2 7 1 7 5 2 6 10 9 3];
poids=[9 9 10 6 4 7 2 8 1 7 1 3 2 9 2 4 3 8 3 4 10 2 2 3 6 5 5 1 10 3];
poidsSac=86;
end
if(kp==6)
profit=[4 3 6 5 9 2 7 7 8 4 7 4 2 10 4 6 4 10 7 8 9 2 4 6 1 8 9 4 10 2];
poids=[3 4 7 8 7 6 9 9 1 8 10 10 9 1 4 4 8 6 6 6 9 7 5 5 7 10 6 4 6 10];
poidsSac=57;
end
if(kp==7)
profit=[3 2 4 5 9 7 8 5 8 3 6 3 8 9 6 9 3 5 2 8 5 10 5 8 8 1 6 1 3 7 7 5 6 6 3 6 2 8
5 4];
poids=[6 8 7 5 1 4 10 10 3 8 10 7 10 8 8 2 4 5 9 1 3 3 8 4 10 6 7 4 9 7 2 1 3 3 2 10
7 3 4 1];
poidsSac=109;
end
if(kp==8)
    profit=[5 3 3 9 9 4 6 4 10 6 3 5 10 5 3 5 5 4 4 8 9 4 4 1 5 1 8 5 2 3 9 4 3 3 2 2
4 2 2 1];
    poids=[1 10 3 3 1 10 2 3 5 10 8 4 7 1 8 3 7 9 7 8 3 5 1 7 5 7 1 2 2 1 3 4 8 10 7
8 7 7 4 2];
```

```
    poidsSac=61;
end
if(kp==9)
profit=[5 1 8 6 10 9 3 7 6 5 6 10 9 1 5 6 2 9 8 6 1 1 8 9 7 6 9 6 8 1 2 4 1 6 8 10 9
9 8 4 3 7 9 5 3 3 5 8 3 7];
poids=[7 10 4 2 6 6 1 7 9 8 6 7 5 1 10 1 7 10 2 3 6 2 1 5 8 8 2 10 1 7 4 3 1 7 6 9 9
10 6 2 6 1 6 1 2 5 2 5 9 3];
poidsSac=146;
end
if(kp==10)
profit=[2 9 8 1 3 2 9 4 5 7 10 9 3 3 7 9 10 4 6 10 3 9 6 8 7 5 1 7 5 10 3 2 2 2 10 7
6 7 2 1 8 4 5 5 1 2 5 6 10 2];
poids=[7 7 2 8 9 6 5 5 1 2 6 10 3 4 3 9 8 10 1 2 10 9 2 2 8 4 4 6 10 9 2 8 7 1 6 6 10
1 6 6 5 5 4 5 1 8 2 3 5 9];
poidsSac=90;
end
coeffTaillePopulationTailleChromosome=a;
coefSelection=c;
coeffCroisement=d;
coeffMutation=e;
nbIterations=f;
nbRepetition=g;
t0=cputime;
matriceSolutions=[];
profitspoidsSolutions=[];
for i=1:nbRepetition
[solution,profitSolution,poidsSolution]=RechercheSolution2(profit,poids,poidsSac,coef
fTaillePopulationTailleChromosome,coefSelection,coeffCroisement,coeffMutation,nbItera
tions);
matriceSolutions=[matriceSolutions;solution];
profitspoidsSolutions=[profitspoidsSolutions;
profitSolution poidsSolution];
end
vecteurProfit=[];
for i=1:nbRepetition
profitSolution=matriceSolutions(i,:)*profit';
vecteurProfit=[vecteurProfit;profitSolution];
end
max=vecteurProfit(1);
iMax=1; for i=1:nbRepetition
if (vecteurProfit(i)>max)
max=vecteurProfit(i);
iMax=i;
end
end
solutionOptimale=matriceSolutions(iMax,:);
profitSolutionOptimale=solutionOptimale*profit';
poidsSolutionOptimale=solutionOptimale*poids';
tempsExecution=cputime -t0; affichObjetSol=[];
for i=1:(length(solutionOptimale))
if (solutionOptimale(i)==1)
affichObjetSol=[affichObjetSol i];
%disp(['Objet : ',num2str(i)])
end
end
%disp(['le temps d''execution est : ',num2str(tempsExecution)])
end
```

ii. Fonctions secondaires

1. La fonction RechercheSolution2 :

   Cette fonction retourne la solution optimale trouvée lors d'une seule exécution de l'algorithme de résolution ainsi que le poids et le profit correspondant.

```
function [solution,profitSolution,poidsSolution]=RechercheSolution2(a,b,c,d,f,g,h,k )
profit=a;
poids=b;
poidsSac=c;
coeffTaillePopulationTailleChromosome=d;
coefSelection=f;
coeffCroisement=g;
coeffMutation=h;
nbIterations=k;
tailleChromosome=length(profit);
%=le nombre d'objets
populationInitiale=CreationPopulationInitiale2(tailleChromosome,coeffTaillePopulation
TailleChromosome);
population=populationInitiale;
poidsSolution=poidsSac+5;
while(poidsSolution>poidsSac)
for i=1:nbIterations
populationApresSelection=SelectionTournoi(population,profit,poids,
poidsSac,coefSelection);
populationApresCroisement=Croisement2(populationApresSelection,coeffCroisement);
populationApresMutation=Mutation2(populationApresCroisement,coeffMutation);
population=populationApresMutation;
end
populationFinale=SelectionTournoi(population,profit,poids,poidsSac,coefSelection)
sizePop=size(populationFinale);
taillePopulationFinale=sizePop(1);
i=1;
sol=populationFinale(i,:)
poidsSol=sol*poids'
while ((i<taillePopulationFinale)&&(poidsSol>poidsSac))
i=i+1;
sol=populationFinale(i,:);
poidsSol=sol*poids';
end
solution=sol;
profitSolution=sol*profit';
poidsSolution=sol*poids';
populationFinaleApresCroisement=Croisement2(populationFinale,coeffCroisement);
population=populationFinaleApresCroisement;
end
end
```

2. La fonction CreationPopulationInitiale2(a,b)

- a= tailleChromosome
- b= coeffTaillePopulationTailleChromosome

→La population initiale est crée aléatoirement.

```
function M = CreationPopulationInitiale2( a, b )
tailleChromosome=a;
coeffTaillePopulationTailleChromosome=b;
taillePopulation=tailleChromosome*coeffTaillePopulationTailleChromosome;
for i= 1:taillePopulation
for j=1:tailleChromosome
if (rand < 0.5)
M(i,j)=0;
else
M(i,j)=1;
end
end
```

```
end
end
```

3. La fonction CalculFitness2(a,b,c, d)

- a=population ,
- b=profit,
- c=poids,
- d=poidsSac

Cette fonction retoune un vecteur comportant les fitness de tous les chromosomes de la population entrée.
Le coefficient de pénalisation= utilité maximale/2
Avec l'utilité maximale est le profit maximale obtenu si tous les objets sont choisis.

```
function [Fit]=CalculFitness2(a,b,c, d)
population=a;
profit=b;
poids=c;
poidsSac=d;
sizePop=size(population);
taillePopulation=sizePop(1);
tailleChromosome=sizePop(2);
utilteMax=profit*ones(tailleChromosome,1);
coefPenalisation=utilteMax/2;
F=[];
for i=1:taillePopulation
poidsChromosome=0;
for j=1:tailleChromosome
poidsChromosome=poidsChromosome+population(i,j)*poids(j);
end
if (poidsChromosome>=poidsSac)
fit=0;
for j=1:tailleChromosome
fit=fit+(population(i,j)*profit(j))-(coefPenalisation*(poidsChromosome-poidsSac));
end
else
fit=0;
for j=1:tailleChromosome
fit=fit+(population(i,j)*profit(j));
end
end
F(i)=fit;
end
Fit=F;
end
```

4. La fonction SelectionTournoi(a,b,c,d,f)

- a= population
- b= profit;
- c=poids;
- d=poidsSac;
- f=coefSelection.

On a testé la sélection roulette wheel (voir Selection2.m puis celle par tournois (voir SelectionTournoi.m).
Cette dernière donne des résultats meilleurs. Les résultats récapitulés au niveau du tableau correspondent à l'utilisation de la selec tion par tournoi.

```
function [populationApresSelection]=SelectionTournoi(a,b,c,d,f)
population=a;
profit=b;
poids=c;
poidsSac=d;
```

```
coefSelection=f;
sizePop=size(population);
taillePopulation=sizePop(1);
tailleChromosome=sizePop(2);
F=CalculFitness2(population,profit,poids, poidsSac);
NP=[];
populationApresSelection=[];
for i=1:(coefSelection*taillePopulation)
indiceChromo1=randi(taillePopulation);
indiceChromo2=randi(taillePopulation);
F1=F(indiceChromo1);
F2=F(indiceChromo2);
if (F1>F2)
p=rand;
if (p <= 0.7)
NP=[NP;population(indiceChromo1,:)];
else
NP=[NP;population(indiceChromo2,:)];
end
else
p=rand;
if (p <= 0.7)
NP=[NP;population(indiceChromo2,:)];
else
NP=[NP;population(indiceChromo1,:)];
end
end
end
populationApresSelection=NP;
end
```

5. La fonction Croisement2(a, b)
   - a= populationApresSelection;
   - b= coeffCroisement

   Pour un nombre de croisements défini par le coefficient de croisement initialement entré dans le Main2 , on choisit `a chaque fois aléatoirement un chromosome Père et un autre Mère et on effectue un croisement à un point dans une position aléatoire.

```
function [ NP ] = Croisement2( a, b )
populationApresSelection=a;
coeffCroisement=b;
sizePop=size(populationApresSelection);
taillePopulation=sizePop(1);
tailleChromosome=sizePop(2);
z=round(taillePopulation* coeffCroisement);
%nombre de croisements
NP=[populationApresSelection];
for i=1:z
indice=i;
indicePere=randi(taillePopulation);
indiceMere=randi(taillePopulation);
Pere=populationApresSelection(indicePere,:);
Mere=populationApresSelection(indiceMere,:);
pointCroisement=randi(tailleChromosome);
Fils1=[Pere(1:pointCroisement),Mere(pointCroisement+1:tailleChromosome)];
Fils2=[Mere(1:pointCroisement),Pere(pointCroisement+1:tailleChromosome)];
NP=[NP;Fils1;Fils2];
end
end
```

5. La fonction Mutation2(a, b)

- a= populationApresCroisement;
- b= coeffMutation

Pour un nombre de mutation défini le coefficient de mutation initialement entré dans le Main2 , on choisit à chaque fois aléatoirement un chromosome qui va subir une mutation au niveau d'un gène choisi aléatoirement.

```
function [ NP ] = Mutation2( a,b )
populationApresCroisement=a;
coeffMutation=b;
sizePop=size(populationApresCroisement);
taillePopulation=sizePop(1);
tailleChromosome=sizePop(2);
nbMutation=round(taillePopulation*coeffMutation);
%nombre de mutation
for i=1:nbMutation
iMutation=randi(taillePopulation);
jMutation=randi(tailleChromosome);
posMutation=[iMutation jMutation];
if (populationApresCroisement(iMutation,jMutation)==0)
populationApresCroisement(iMutation,jMutation)=1;
else
populationApresCroisement(iMutation,jMutation)=0;
end
end
NP=populationApresCroisement;
end
```

Conclusion

Malgré l'essor que connait l'optimisation combinatoire ces dernières décennies, il n'existe pas d'algorithme de résolution polynomial pour certains problèmes. Ceux-ci consiste en la recherche d'un élément de meilleure valeur dans un ensemble fini mais de trop grande cardinalité.

La résolution de ces problèmes d'optimisation combinatoire NP-difficiles passe alors par des méthodes approchées ou dites méta-heuristiques.

Dans la famille des techniques approchées d'optimisation existantes, les algorithmes génétiques se sont imposés comme des heuristiques performantes.

Afin d'augmenter les performances des algorithmes génétiques, il est apparu souhaitable de doter un AG d'une très grande population. Toutefois, lorsque la taille de la population est élevée le temps de convergence du AG augmente au-delà du raisonnable. Il devient nécessaire de paralléliser le calcul.

Exécution :

Instance	Nombre d'objets	Poids max du sac	Poids de la solution optimale	Objets sélectionnés	Profits de la solution optimale	Paramètres d'entrés
KP1	10	27	26	7, 1, 5, 8, 3, 2, 6, 10	49	Main2(1,3,0.5,0.5,0.05 ,100,5)
KP 2	10	20	20	4, 10, 5, 6, 7	31	Main2(2,3,0.5,0.5,0.05 ,100,7)
KP 3	20	58	58	20, 15, 10, 5, 16, 12, 4, 14, 1, 18, 17, 8, 13, 11	99	Main2(3,3,0.5,0.5,0.05 ,100,8)
KP 4	20	38	38	8, 20, 13, 11, 7, 6, 9, 16, 17, 12, 15	83	Main2(4,3,0.5,0.5,0.05 ,100,5)
KP 5	30	86	86	29, 28, 30, 27, 15, 24, 10, 2, 11, 12, 5, 19, 22, 4, 6, 9, 20, 17, 8, 13, 7	143	Main2(5,3,0.5,0.5,0.05 ,100,2)
KP 6	30	57	57	27, 1, 14, 5, 9, 2, 24, 28, 15, 29, 20, 16, 18	91	Main2(6,3,0.5,0.5,0.05 ,100,10)
KP 7	40	109	109	24, 38, 40, 32, 9, 5, 20, 16, 25, 6, 7, 22, 15, 13, 18, 35, 14, 30, 31, 21, 33, 17, 27, 39, 34	168	Main2(7,3,0.5,0.5,0.05 ,100,2)
KP 8	40	61	61	1,3,4,5,7,8,9,12,13,14 ,16,21,23, 25,27,28,29,30,31,32, 39,40	124	Main2(8,5,0.5,0.5,0.05 ,100,20)
KP 9	50	146	146	3,4,5,6,7,8,11,12,13,16,18,19,20 ,23,24,25,26,27,29,32,33,35,36, 37,39,40,42,43,44,45, 46,47,48,5 0	235	Main2(9,6,0.5,0.5,0.05 ,100,20)
KP 10	50	90	90	2,3,7,9,10,11,15,19 ,20, 23,24,26,28,30,34,35, 36,38,41,4 3 ,44,47,48,49	49	Main2(10,6,0.5,0.5, 0.0 5,100,20)