Homework 3 : Simulated Annealing and Genetic Algorithm

University of Nantes

Master 1 Optimization in operational research

Class of Metaheuristics

Professor : Xavier Gandibleux

UNIVERSITÉ DE NANTES

Léon Ferrari

University of Nantes

Master 1 Optimization in operational research

Written with LaTeX

November 3, 2017

Contents

1 Introduction to the Set Packing Problem **3**
1.1 JuMP modelisation for the set packing problem 3
1.2 Ten instances of the SPP 4
1.3 Constructing a solution 4
1.3.1 Greedy randomized adaptive search procedure (GRASP) 4
1.3.2 Tutorial step by step 5
1.3.3 Reactive GRASP 7
1.3.4 Feasibility of our solution 8
1.4 Improving our solution 9
1.4.1 Simulated annealing 9
1.4.2 A greedy local search 10
1.5 Results and performances 10
1.5.1 System and environment 10
1.6 Interpreting the results 12

1 Introduction to the Set Packing Problem

The Set Packing Problem with a finite size n of item and m of constraints.
It can be define as follow, with :

$$\begin{array}{lll} Max\, Z & = & \sum_{i=1}^{n} x_i \times c_i \\ s.c. & \sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j} \times x_i & \leq \quad 1 \\ x_i \in \{0,1\} & & \end{array}$$

With :

$$x_i \quad where \left\{ \begin{array}{ll} 1 & if\, x_i \in solution. \\ 0 & otherwise \end{array} \right\}$$

$$x_{i,j} \quad where \left\{ \begin{array}{ll} 1 & if\, x_{i,j} \in constraint\, i. \\ 0 & otherwise \end{array} \right\}$$

$i \in [1, n], \quad j \in [1, m]$
x_i a variable of the solution.
c_i the cost of variable i.
$x_{i,j}$ an element of the constraint matrix.
This type of problem can be illustrated in several ways, I will introduce you to some of them.

Illustration 1 :
You are moving out of your apartment and you need to pack your different item.
Obviously some of theses objects are more important for you or more expensive.
But you do not have enough boxes so you will have to choose which one you will protect from a possible break.
Here the coefficient of the objective value will be the value of the different items and the constraints will represent the boxes that you own.

Illustration 2 :
You are a manager and you have to create a sport team for your company. In order to select them you send them to pass different test.
You also asked them with who they don't want be if they are in the sport team.
Here the coefficient of the objective value will be the result of the test and the constraints will represent the people they don't want to be with.

1.1 JuMP modelisation for the set packing problem

The modelling of the problem have been done with the language Julia and the package JuMP (Julia for mathematical programming). We used the GLPKSolverMIP , the GLPK solver for mixed-integer problem. Once the data describing the problem have been extract from the file, we can create a model with JuMP and send it to GLPK in order to solve it with the following instructions :

Listing 1: Configuration du demon SNMP

```
#= This is a code modelling and solving a SPP problem with JuMP and GLPK)
model  = Model(solver=GLPKSolverMIP())
```

```
@variable( model,  x[1:n], Bin)
@objective( model , Max, sum( Variables[j] * x[j] for j=1:n ) )
@constraint( model , cte[i=1:m], sum(Constraints[i,j]* x[j] for j=1:n)
<= 1 )
solve(model)
```

1.2 Ten instances of the SPP

Problem Name	Number of Variables	Number of constraints	Best known value
100rnd0100.dat	100	500	372
100rnd0200.dat	100	500	34
100rnd0300.dat	100	500	203
100rnd0400.dat	100	500	16
100rnd0500.dat	100	100	639
100rnd0600.dat	100	100	64
100rnd0700.dat	100	100	503
100rnd0800.dat	100	100	39
100rnd0900.dat	100	300	463
100rnd1000.dat	100	300	40

1.3 Constructing a solution

1.3.1 Greedy randomized adaptive search procedure (GRASP)

In order to build different good solution we will use the construction algorithm called GRASP from Mauricio G. C. Resende. It is an easy to program and understand algorithm. The aim is too pick randomly the candidate which will be set to one through a random candidate list. The major interest of this algorithm is the technique used create the random candidate list therefore a parameter, α, is used to built it. α is a coefficient used in the creation of the random candidate list, it will be used to compute a lower bound limit for the selection of the candidate variable.

Once we have compute the utility for each variables, in the case of the SPP problem the so called "utility" is :

$$U_i = \frac{c_i}{\sum_{j=1}^{m} x_{j,i}}$$

In the following steps *Utility* will be considered as a Vector of size n containing the utility of each variable. The lower bound limit for the utility will be compute as follow :

$$Limit = \min(Utility) + \alpha \times (\max(Utility) - \min(Utility))$$

And now the construction algorithm :

Data: Alpha,V,C
Result: Solution
/* C the constraint matrix, V the variables cost */
/* Sort the array in decreasing order based on the second row */
Utility = $\forall v \in V, U_v = \frac{Cost_v}{\sum_{j=1}^{m} X_{j,v}}$
/* Sort by decreasing order */
sort(Utility)
while *NotEmpty(RCL)* **do**
  Limit = Min(Utility) + Alpha * (Max(Utility)- Min(Utility))
  RCL = ∅
  for *i := 1 to N* **do**
    if *Utility[i] >= Limit* **then**
      RCL = add(RCL,i)
    else
      break
    end
  end
  Candidate = RandomSelect(RCL)
  UpdateUtility(Utility)
  addToSolution(Solution,Candidate)
end
return Solution

Algorithm 1: GRASP Construction

I will show step by step how this algorithm works with this example :

$$\begin{array}{lrcl} Max\, Z & = 38 \times X_1 + 26 \times X_2 + 17 \times X_3 + 12 \times X_4 + 11 \times X_5 & & \\ s.c. & X_1 + X_3 & \leq & 1 \\ & X_2 + X_4 & \leq & 1 \\ & X_1 + X_5 & \leq & 1 \\ & X_1 + X_2 & \leq & 1 \end{array}$$

$X_i \in \{0, 1\}$

1.3.2 Tutorial step by step

I will show step by step how this algorithm works with this example, here we have 5 variables and 4 constraints. So m and n will take the value of 5 and 4, respectively.
I will now compute the utility of each variables, that can be expressed as follows for variable i :

$$U_i = \frac{c_i}{\sum_{j=1}^{n} x_{i,j}}$$

So we have the two dimension array "Utility" like this :

Utility =

1	2	3	4	5
12, 66	12.5	17	12	8

Table 1: GRASP construction : Step 1

$$
\begin{aligned}
Solution &= \emptyset \\
Limit &= \min(Utility) + \alpha \times (\max(Utility) - \min(Utility)) \\
Limit &= 8 + 0.75 * (17 - 8) \\
Limit &= 14.75 \\
RCL &= \{3\} \\
Candidate &= rand(RCL) \\
Candidate &= 3 \\
Solution &= \{3\}
\end{aligned}
$$

$$
\text{Utility} = \begin{array}{|c|c|c|} \hline 2 & 4 & 5 \\ \hline 12.5 & 12 & 8 \\ \hline \end{array}
$$

Table 2: GRASP construction : Step 2

$$
\begin{aligned}
Solution &= \{3\} \\
Limit &= \min(Utility) + \alpha \times (\max(Utility) - \min(Utility)) \\
Limit &= 8 + 0.75 * (12.5 - 8) \\
Limit &= 11.375 \\
RCL &= \{2, 4\} \\
Candidate &= rand(RCL) \\
Candidate &= 2 \\
Solution &= \{3, 2\}
\end{aligned}
$$

$$
\text{Utility} = \begin{array}{|c|} \hline 5 \\ \hline 8 \\ \hline \end{array}
$$

Table 3: GRASP construction : Step 3

$$
\begin{aligned}
Solution &= \{3, 2\} \\
Limit &= \min(Utility) + \alpha \times (\max(Utility) - \min(Utility)) \\
Limit &= 8 + 0.75 * (8 - 8) \\
Limit &= 8 \\
RCL &= \{5\} \\
Candidate &= rand(RCL) \\
Candidate &= 5 \\
Solution &= \{3, 2, 5\}
\end{aligned}
$$

The RCL is now finish and we have the solution {3,2,5} with an objective value worth 37.5.

1.3.3 Reactive GRASP

Reactive Grasp is just a probabilistic system that allow you to auto adjust(auto-tune) the α parameter given the results obtained.
Given AlphaSet, a finite set of α of size N with $\forall \alpha_i \in AlphaSet, \alpha_i \in \mathbb{R}, \alpha_i \in [0,1]$, and AlphaProba a finite set of uniformly distributed probability of size N with $\forall p_i \in AlphaProba, p_i = \frac{1}{N}$
At first every α_i have the same probability p_i to be picked, the objective of reactive Grasp is to adjust every probability after a given number of run in order to increase the probability p_i of an α_i giving good result.
The new probability can be determine with the following formula :

$$k_i = \frac{Average(Z_{p_i}) - Min(Z)}{Max(Z) - Min(Z)}$$

$$p_i = \frac{k_i}{\sum_{j=1}^{N} k_j}$$

Where Z is the objective value obtained by a GRASP construction followed by a local search. Here the maximum and the minimum are global over the N run without any discrimination based on α. Only the average is computed with respect to the α that have been used for the construction.

```
/* AlphaProba :  a vector containing the possibillity for each α.      */
/* Min,Max,Average are the min,max value found.  N the number of α.    */
/* Average a vector containing the man value for each α used.          */
Data: AlphaProba,Min,Max,Average, N
Result: AlphaProba
for j := 1 to N do
 | NewValue[j] = ( Average[i] - Worst ) / ( Max - Worst )
end
SumOfNew = sum(NewValue)
for i := 1 to N do
 | AlphaProba[i] = NewValue[i] / SumOfNew
end
return AlphaProba
```

Algorithm 2: UpdateReactiveGRASP

```
/* N the number of α.                                                  */
/* AlphaProba :  a vector containing the possibillity for each α.      */
/* AlphaVal :  a vector containing the value of each α.                */
Data: AlphaProba,AlphaVal,N
Result: Index,Alpha
Proba = random() Val = 0 for j := 1 to N do
 | val += AlphaProba[j] if Proba <= Val then
 |  | return j,AlphaVal[j]
 | end
end
return N,AlphaVal[N]
```

Algorithm 3: ReactiveGRASP

1.3.4 Feasibility of our solution

No journey have been took out of the feasible region of the solutions. I will introduce you how I manage to respect the consistency of a problem and so respect the feasibility of any solution found.
For this purpose I use two vector :
RM a vector of size n, to store the current state of all the constraints(Saturated or not).
Freedom, a vector of size m, to store the current state of all the variables(Free or not).

$\vec{v_{RM}}$:

$$\forall rm_i \in \vec{v_{RM}} :$$

$$rm_i = \left\{ \begin{array}{ll} 1 & if\ \exists x \in x_{i,:} = 1 \\ 0 & otherwise \end{array} \right\}$$

$\vec{v_{freedom}}$:

$$\forall x_i \in \vec{v_{freedom}} :$$

$$x_i = \sum_{j=1}^{m} -x_{j,i}$$

Freedom is a vector full of zeros at the begin. When a variable x is set to one all the other variables that are present in the constraint where x is are decremented.
When a variable x is set to zero all the other variables that are present in the constraint where x is are incremented. So a variable is free when its value in the vector is equal to zero.
The RM vector is mainly used to gain speed as the Freedom vector, moreover this last one allow us to know how many constraints are "blocking" a given variable to be set to one.

```
/* CurrentUsedVar :  a vector containing the variable used,X : the index
   of the var to set to one                                              */
/* Freedom :  the current state of the variables,Cost :  the vector of
   cost, Solution :  the current solution                                */
/* Constraints :  the constraint matrix,RMconstraints :  the current state
   of the constraints                                                    */
Data: Constraints,Freedom,Cost,RMConstraints,Solution,CurrentUsedVar, X
Result: Boolean,Solution
/* X is the var to set to one, and                                       */
if Freedom[X] == 0 and Solution[X] == 0 then
    for j := 1 to M do
        if Constraints[j,X] == 1 then
            if RM[j] == 0) then
                RM[j] = 1
                for i := 1 to N do
                    if Constraints[j,i] == 1 then
                        Freedom[i]-=1
                    end
                end
            else
                return False,CurrentSolution
            end
        end
    end
else
    return False,CurrentSolution
end
Solution[X] = 1
CurrentUsedVar = add(CurrentUsedVar,x)
ObjectiveValue +=Cost[X]
return True,Solution
```

Algorithm 4: SetToOne

1.4 Improving our solution

I had a few different algorithm in order to improve my solution. One composed of Simulated Annealing using to different move, an AddElseDrop and a Drop1AddMax.
And another one based on a kpexchange, setting 1 to 3 to zero and a maximum number to one.

1.4.1 Simulated annealing

I tried Simulated Annealing but because of the random move it operate it does not allow us to use GRASP construction at it's full potential. Why ? Because Reactive Grasp is a short term memory used at the same time as GRASP construction to auto-tune the α parameter for a given problem. If you use an algorithm like SA[1] that doesn't take advantage of a good construction but build and improve it's solution based on dubious choice for the main part of it's run. So the probability compute by reactive GRASP will be biased.

I will just show you some result I obtained and compare them with kp exchange.

[1] Simulated Annealing

1.4.2 A greedy local search

I used a 22-exchange as greedy local-search.

1.5 Results and performances

1.5.1 System and environment

We solved the problem on the following system :

OS	Centos 7 64 bits
Processor	i7 4720HQ
CPU Freq	2.6GHz
Physical Core CPU	4
RAM	8GB
Julia	Version 0.6.0
Solver	GLPK

We obtained the following result :

Problem	GLPK		Heuristic construction		Local search	
Data	$\tilde{Z}$	Time(s)	$\tilde{Z_{avg}}$	Time(s)	$\tilde{Z_{avg}}$	Time(s)
100rnd0100	372	2.302	338.755	0.0001	352.01	0.053
100rnd0200	34	3.653	29.675	0.0001	33.985	0.0584
100rnd0300	203	0.574	181.265	0.0555	185.81	0.0630675
100rnd0400	16	3.937	13.295	0.0606	15.31	0.07259
100rnd0500	639	0.000623319	617.24	0.0001	621.435	0.0226125
100rnd0600	64	0.000717666	61.5	0.0235	64	0.0067
100rnd0700	503	0.000726351	491.715	0.0001025	497.7	0.0247024
100rnd0800	39	0.157	35.99	0.0437	38.32	0.02034
100rnd0900	463	0.151	431.85	0.0001	446.84	0.037997
100rnd1000	40	0.422	36.17	0.0001	39.525	0.0402

We are gonna compare two problems, the *100rnd0100.dat* and the*1000rnd0100.dat*.
Best value known : 372
Let's have a closer look to 200 runs of our algorithm on the problem *100rnd0100.dat*.
I am now going to introduce the result found every 20 runs, there is going to be the average the minimum and the maximum. Result every 20 iterations :

	22Exchange			Simulated annealing		
Runs	$\tilde{Z_{min}}$	$\tilde{Z_{avg}}$	$\tilde{Z_{max}}$	$\tilde{Z_{min}}$	$\tilde{Z_{avg}}$	$\tilde{Z_{max}}$
20	308	335	365	338	351	365
40	305	338	356	328	352	372
60	298	336	363	331	351	367
80	313	335	362	331	351	372
100	300	339	368	329	350	369
120	313	338	365	338	350	365
140	308	336	362	328	351	369
160	274	338	367	326	353	369
180	314	337	368	330	350	364
200	291	338	368	338	351	365

And here how the α changed :

Before runs	α value	0.5	0.6	0.75	0.9
	α probability	0.25	0.25	0.25	0.25
After runs	α with KPexchange	0.264422	0.228431	0.273098	0.234049
	α with CS	0.246867	0.247649	0.27542	0.230065

By looking at the result we observe no big difference except for the average value of $\tilde{Z}$ that is bigger with Simulated Annealing than with a classic local search. But what is more interesting is the maximum found with the KP exchange, it is higher than the one found with CS. Even if it is not always the case it is important to notice it.

Let's have a closer look to 200 runs of our algorithm on the problem *1000rnd0100.dat* :
Best value known : 67
I am now going to introduce the result found every 20 runs, there is going to be the average the minimum and the maximum. But this time we only KP-Exchange. Result every 20 iterations :

	22Exchange		
Runs	$\tilde{Z_{min}}$	$\tilde{Z_{avg}}$	$\tilde{Z_{max}}$
20	26	38	57
40	26	38	57
60	26	39	56
80	24	39	59
100	25	39	57
120	24	38	57
140	25	40	56
160	25	40	55
180	27	40	57
200	27	39	59

And here how the α changed :

Before runs	α value	0.5	0.6	0.75	0.9
	α probability	0.25	0.25	0.25	0.25
After runs	α with KPexchange	0.180245	0.210958	0.278561	0.330236

Here we can notice a pretty big difference in the probability. I notice that the probability really do change for large problem. Maybe because there are more candidates to be in a RCL for a given alpha so there is a real difference between the RCL of an $\alpha = 0.5$ and an $\alpha = 0.8$. I noticed the same with other large problem like the *2000rnd0100.dat*.

1.6 Interpreting the results

But in this document we just see the use of meta-heuristics on reasonably small problem, I have been running my heuristics on bigger problems and it was way faster. Nevertheless I didn't include them in this report because I forget to record them.

The construction of a good feasible solution is the part of the algorithm that get the solution really close to the best one. However, having a multiple start solution allow us to visit a bigger neighbourhood. The construction's heuristics is a good way to join the edge of the best solution, but it can also be the one that discard you from the best solution by making a bad choice from the beginning. Therefore questioning the construction algorithm in the local search's heuristics is a good idea but in our case here, because we have a multi start algorithm, it could be more logical to be greedy and simultaneously try to thin our neighbourhood search scope. Don't take it for granted.

If we compare the time spend by the meta-heuristics and the MIP Solver it appears that on problem with a large number of constraints the heuristics have a really good score compare to the solver MIP. And there is also a really small difference between the value found by the heuristics and the value found by the solver.

I think that there are many ways to improve my algorithm, and I improved it many time by the end of the time imparted. Here is what I changed :

1. I kicked out of the problem the variables that are not in any constraints. BY setting them to one without having any possibility to set them to 0.

   Maybe I could also delete the row in the constraints matrix and everywhere in the problem but it look pretty difficult.

2. Therefore I noticed that we can order the array containing the utilities. so we can stop going through it when building a solution with GRASP.

   I now store an array containing the value of the current var. So no need to seek them in the full variables array every-time I need to know which one is free.

3. I noticed that I don't need to update a constraint matrix copy to see which constraints is overload or which variables is not free.

And many other changes..

To make a better heuristic I think we need :

1. To learn some maths tricks to simplify some type of problem.

   The ones who are not obvious...

2. To study more deeply a type of problem and data-structures that fit it.

3. To spend more time studying heuristics in order to see from a different prism this type of problem.