GRASP for Set Packing Problems

Xavier DELORME[1,2], Xavier GANDIBLEUX[2] and Joaquin RODRIGUEZ[1]

Abstract— **In this paper, we describe two implementations of the metaheuristic GRASP (Greedy Randomized Adaptative Search Procedure) for the Set Packing Problem (SPP). One of them is customized for a particular Unicost SPP which describes a real problem. Its main characteristics are presented and their influence on GRASP running is commented on. Results observed on several instances of this problem are reported and compared with Cplex results. At the end, some improvement tracks are mentionned.**

Keywords— **Metaheuristic GRASP, Set Packing Problem**

I. Introduction

This work concerns the resolution of the "set packing problem" (SPP). It is a classic optimization problem, close to the "set covering problem" (SCP) [1], [2]. It could be reformulated as a "node packing problem" (NPP). Strangely the SPP has not received a lot of attention. In comparison with the SCP and the NPP, very few works concern the SPP resolution.

We are investigating the resolution of this combinatorial problem because we are working on a real railway problem which can be formulated as an SPP [3], [4]. The main questions deal with the infrastructure capacity evaluation of a railway network. The numerical instances available represent the Gonnesse crossing point, north of Paris.

In a first step the resolution has been achieved using Cplex [5]. Nevertheless, the size of numerical instances quickly becomes large, compromising the exact resolutions with Cplex with reasonable solving time. In a second step, a heuristic method derived from the GRASP (Greedy Randomized Adaptative Search Procedure) metaheuristic [6] has been developped. Two general operational procedures, which can solve any set packing problem (SPP), have been developped. One of them, the most efficient on the type of instances of our railway problem, has also been customized to take into account the specific structure present in this problem.

In this paper, we present the two GRASP algorithms for solving the SPP. Section II introduces the SPP in the general case and the specific properties of our numerical instances. The principles of GRASP are reminded in section III, with the algorithmic description of both our operational procedures. Section IV reports and comments the numerical results collected. A conclusion is given in section V.

[1] INRETS-ESTAS 20 rue Elisée Reclus, F59650 Villeneuve d'Ascq – France e-mail: {Xavier.Delorme,Joaquin.Rodriguez}@inrets.fr
[2] LAMIH – UMR CNRS 8530, Université de Valenciennes, Campus "Le Mont Houy", F-59313 Valenciennes cedex 9 – France e-mail: Xavier.Gandibleux@univ-valenciennes.fr

II. Set Packing Problems

A. The general SPP formulation

Given a set I of valuated items, the objective of the Set Packing Problem is to select items, taking into account incompatibilities between items, so that the total value of selection is maximized. This problem can be formulated as a mathematical model (1) :

$$\left[\begin{array}{ll} Max\ z = \sum_{i\in I} c_i x_i & \\ \qquad \sum_{i\in I} t_{i,j} x_i \leq 1, \forall j \in J & \\ \qquad x_i \in \{0,1\} & ,\forall i \in I \\ \qquad t_{i,j} \in \{0,1\} & ,\forall i \in I, j \in J \end{array}\right] \qquad (1)$$

considering :

- a vector $x = (x_i)$ where $x_i = \begin{cases} 1 \text{ if item } i \text{ is selected} \\ 0 \text{ otherwise} \end{cases}$
- a vector $c = (c_i)$ where $c_i =$ value of the item i
- a matrix $t = (t_{i,j})$ where $t_{i,j} = \begin{cases} 1 \text{ if item } i \text{ is concerned} \\ \quad \text{by incompatibility } j \\ 0 \text{ otherwise} \end{cases}$

This problem is known to be NP-Hard [1], [2]. The best exact method known for this problem is a Branch & Cut method using polyhedral methods in order to obtain facets, essentially with the determination of cliques [7]. However, only small-sized instances can be solved exactly.

Strangely, very few applications or papers are reported. Nevertheless, we can note two applications of the SPP formulation. First, Mingozzi and al. [8] used it in order to calculate bounds for RCPSP (Resource Constrained Project Scheduling Problem) with a greedy method. Second, Zwaneveld et al. [9] formulated a real railway feasibility problem as a SPP and solved it exactly with some reduction tests and a Branch & Cut method.

The Node Packing Problem (2) is a particular case of the Set Packing Problem in which there are only incompatibilities between two items :

$$\left[\begin{array}{l} Max\ z = \sum_{i\in I} c_i x_i \\ \qquad x_i + x_j \leq 1, \forall (i,j) \text{ incompatible} \\ \qquad x_i \in \{0,1\}\ ,\forall i \in I \end{array}\right] \qquad (2)$$

All Set Packing Problems can be formulated as a Node Packing Problem where each incompatibility is splitted into some incompatibilities between two items. However this reformulation increases the number of incompatibilities ; each constraint between n items is rewritten in C_n^2 constraints between 2 items. Also, a Node Packing Problem (but not directly a Set Packing Problem) can be expressed as a Set Covering Problem with a simple variable modification ($x'_i = 1 - x_i, \forall i \in I$).

B. SPP particularities

The main questions of our problem deal with the infrastructure capacity evaluation of a railway network. Its formulation is inspired by the railway feasibility problem formulation of [9]. A more complete description of this problem is available (see [3], [4]). It introduces some particularities with regard to the general SPP. First, it concerns the variables. Each x_i represents a train on a route. As no preference is expressed, $c_i = 1, \forall i \in I$ defining an Unicost SPP. Second, the constraints have a physical sense, highlighting two particular subsets of constraints :

(a) Given n subset $(I_1, ..., I_n) \in I$, $\bigcap_{k=1}^{n} I_k = \emptyset$ and $\bigcup_{k=1}^{n} I_k = I$ then all items which are in the same subset are incompatible. These constraints concern all items once and once only and thus are unconnected.

(b) Given a matrix $l_{i,j} = \begin{cases} 1 \text{ if item } i \text{ and } j \text{ are incompatible} \\ 0 \text{ otherwise} \end{cases}$, some items from a subset can be incompatible with items from an other subset according to values in matrix l. These constraints are thus connected.

Finally, our particular USPP (3) is formulated as follows :

$$\left[\begin{array}{lll} Max\ z = \sum_{i \in I} x_i & & \\ \quad \sum_{i \in I_k} x_i \leq 1 & , \forall k \in 1..n & (a) \\ \quad x_i + \sum_{j \in I_k} l_{i,j} x_j \leq 1, \forall i \in I, k \in 1..n, i \notin I_k & & (b) \\ \quad x_i \in \{0, 1\} & , \forall i \in I & \end{array}\right] \quad (3)$$

Three consequences are mentioned from these particularities :

1. According to constraints (a), we know that

$$z = \sum_{i \in I} x_i = \sum_{k=1}^{n} \sum_{i \in I_k} x_i \leq n$$

2. If $n > 1$, there are more constraints than variables and more (b) type constraints than (a) type constraints :
- (a) type constraints number is equal to n
- (b) type constraints number, according to matrix $l_{i,j}$, can be equal to $card(I) * (n - 1)$ because if $\exists i \in I$ and $k \in 1..n, i \notin I_k, l_{i,j} = 0\ \forall j \in I_k$ then there is a constraint $x_i \leq 1$ which can be not considered

3. The density of the constraint matrix decreases when the subset number increases :
- (a) type constraints concern only one subset
- (b) type constraints concern only two subsets

A didactic example of the constraint matrix in a case with 15 variables distributed in 6 subsets is presented in Figure 1.

(a)	1	1	1												
				1	1										
						1	1								
								1	1	1					
											1	1			
													1	1	1
(b)	1			1	1										
	1					1	1								
		1		1	1										
		1				1	1								
			1	1	1										
					1	1	1								
						1		1	1						
							1			1					
							1						1	1	1
								1			1				
									1		1				
												1	1	1	1

Fig. 1. A didactic example of constraint matrix

III. Greedy Randomized Adaptative Search Procedure

Due to the important size of considered instances, we used an adaptation of the metaheuristic GRASP (Greedy Randomized Adaptative Search Procedure). This is a multistart two-phase metaheuristic for combinatorial optimization proposed by Feo and Resende [6].

First, a construction phase builds an initial solution with a greedy randomized procedure. This random character enables to obtain solutions in different areas of admissible solution space. Second, a local search phase improves these solutions. This process is repeated many times in order to compensate the random character of the greedy phase (see Figure 2). Several new components extend the scheme of GRASP. They are presented and discussed in [10].

It is easy to customized this metaheuristic on any problems for which construction and local search algorithms are available. GRASP has been applied to a wide range of optimization problems. These include academic and industrial problems in scheduling, routing, logic, partitioning, location and layout, graph theory, assignment, manufacturing, transportation, telecommunications, electrical power systems, and VLSI design. An extensive anotated bibliography is available (see [11]).

Some papers have shown that the GRASP method produces good quality solutions for hard combinatorial optimization problems, particularly for the set covering [12], [13], [14], [15], the node packing [13], [16] and the set packing problems [3], [4].

So, an implementation of GRASP for a specific problem implies choices for four main parameters :
- The greedy method
- The random character importance which is fixed by a parameter $\alpha \in [0; 1]$
- The neighbourhood considered for local search
- The stopping criterion

We propose two different implementations of GRASP for

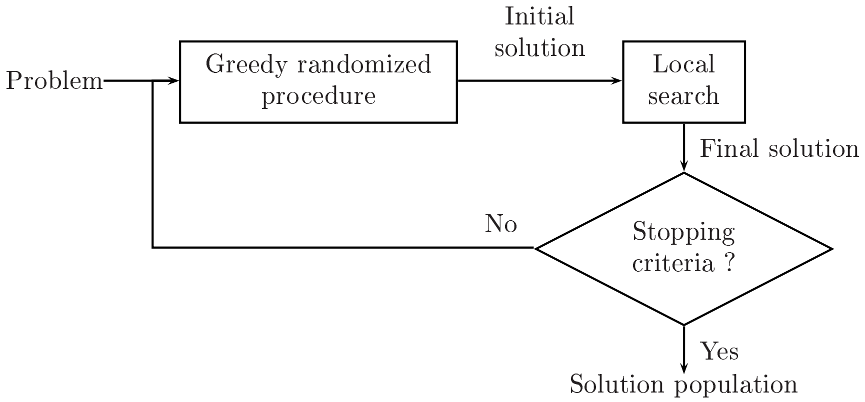

Fig. 2. The GRASP process

a generic Set Packing Problem. Due to their different philosophy, they are not efficient on the same instances.

A. The first implementation of GRASP for general SPP

The first implementation is inspired by some implementations [12], [14], [15] of a GRASP algorithm for Set Covering Problem [13]. It is more efficient when the optimal solution is important compared to the variable number.

A.1 The greedy randomized phase

The greedy procedure (Algorithm 1) builds a solution from the trivial non-admissible solution of maximal value ($x_i = 1, \forall i \in I$). Some variable values are changed (ie fixed to 0) until we obtain an admissible solution. These changes concern only one variable per iteration. In order to keep a maximal value for the objective function, variables which concern a maximum number of constraints and with a minimum value (ie with c_i minimum) are prioritized, but choice is random among the most interesting variables (Restricted Candidate List).

$x_i \leftarrow 1, \forall i \in I$
while $(j \neq \emptyset)$ loop
  $Eval_i \leftarrow \sum_{j \in J} t_{i,j}/c_i, \forall i \in I$
  $Limit \leftarrow \alpha * \max_{i \in I}(Eval_i)$
  $RCL \leftarrow \{i \in I, Eval_i \geq Limit\}$
  $i^* \leftarrow RandomSelect(RCL)$
  $x_{i^*} \leftarrow 0$
  $I \leftarrow I \setminus \{i^*\}$
  $J \leftarrow J \setminus \{j \in J, \sum_{i \in I} t_{i,j} x_i \leq 1\}$
endWhile

Alg. 1. The first greedy randomized algorithm

An exemple of a feasible greedy phase for the didactic exemple with $\alpha = 0.8$ is presented in Figure 3.

$step\ 1:\ Eval \leftarrow \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 3&3&2&4&5&5&6&3&3&2&3&2&3&3&3\\\hline\end{array}$
$Limit \leftarrow 4.8 \Rightarrow RCL \leftarrow \{5,6,7\}$
$i^* \leftarrow 6 \Rightarrow x_6 \leftarrow 0$
$J \leftarrow \{1,2,4,5,6,7,8,9,10,11,12,13,14,15\}$
$step\ 2:\ Eval \leftarrow \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 3&3&2&4&5&-&5&3&3&2&3&2&3&3&3\\\hline\end{array}$
$Limit \leftarrow 4.0 \Rightarrow RCL \leftarrow \{4,5,7\}$
$i^* \leftarrow 5 \Rightarrow x_5 \leftarrow 0$
$J \leftarrow \{1,4,5,6,7,8,9,10,11,13,14,15\}$
$\vdots \qquad (x_{7,11,14,4,15,9,12,2,10} \leftarrow 0)$
$step\ 12:\ Eval \leftarrow \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|}\hline 1&-&1&-&-&-&-&-&-&-&-&-&-&-&-\\\hline\end{array}$
$Limit \leftarrow 0.8 \Rightarrow RCL \leftarrow \{1,3\}$
$i^* \leftarrow 1 \Rightarrow x_1 \leftarrow 0$
$J \leftarrow \emptyset$

Fig. 3. A first greedy phase for the didactic exemple

A.2 The local search phase

The neighbourhood used for local search procedure is $k-p$ exchanges. A $k-p$ exchange consists in fixing to 0 of k variables and to 1 of p others variables. Due to the combinatorial explosion of the number of exchange possibilities when k and p increase, we are obliged to limit them. So we test $0-1$ exchanges, $1-1$ exchanges, $2-1$ exchanges and $1-2$ exchanges (Algorithm 2). We only accept exchanges increasing the objective function. When an exchange is accepted, all exchange possibilities are tested again. Local search stops when there is no more possible exchange.

$I_1 \leftarrow \{i \in I, x_i = 1\}$
$J_1 \leftarrow \{j \in J, \exists i \in I_1, t_{i,j} = 1\}$
for $i_1 \in I_1$ loop
  $J_2 \leftarrow J_1 \setminus \{j \in J_1, t_{i_1,j} = 1\}$
  $I_2 \leftarrow \{i \in I, \forall j \in J_2, t_{i,j} = 0\}$
  for $i_2 \in I_2$ loop
    if $\exists i_3 \in I_2, c_{i_1} < c_{i_2} + c_{i_3}$
      and $\forall j \in J_2, t_{i_2,j} + t_{i_3,j} = 0$
      and $\forall j \in J \setminus J_2, t_{i_2,j} + t_{i_3,j} \leq 1$ then
        $x_{i_1} \leftarrow 0$
        $x_{i_2} \leftarrow 1$
        $x_{i_3} \leftarrow 1$
        return()
    endIf
  endFor
endFor

Alg. 2. The 1-2 exchange algorithm

An exemple of local search phase for the greedy solution generated in the previous exemple is presented in Figure 4.

$I_1 \leftarrow \{3, 8, 13\}$
$J_1 \leftarrow \{1, 11, 4, 13, 16, 6, 15, 18\}$
step 1 : $i_1 \leftarrow 3$
$J_2 \leftarrow \{4, 13, 16, 6, 15, 18\}$
$I_2 \leftarrow \{1, 2, 4, 5\}$
| $i_2 \leftarrow 1, \nexists i_3$
| $i_2 \leftarrow 2, \nexists i_3$
| $i_2 \leftarrow 4, \nexists i_3$
| $i_2 \leftarrow 5, \nexists i_3$
step 2 : $i_1 \leftarrow 8$
$J_2 \leftarrow \{1, 11, 6, 15, 18\}$
$I_2 \leftarrow \{6, 9, 10, 11\}$
| $i_2 \leftarrow 6, \exists i_3 = 10$
| $x_8 \leftarrow 0, x_6 \leftarrow 1$ and $x_{10} \leftarrow 1$

Fig. 4. A local search phase for the didactic exemple

$Eval \leftarrow \boxed{3}\boxed{3}\boxed{2}\boxed{4}\boxed{5}\boxed{5}\boxed{6}\boxed{3}\boxed{3}\boxed{2}\boxed{3}\boxed{2}\boxed{3}\boxed{3}\boxed{3}$
step 1 : $Limit \leftarrow 2.4 \Rightarrow RCL \leftarrow \{3, 10, 12\}$
$i^* \leftarrow 3 \Rightarrow x_3 \leftarrow 1$
$I \leftarrow \{6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$
step 2 : $Limit \leftarrow 2.4 \Rightarrow RCL \leftarrow \{10, 12\}$
$i^* \leftarrow 12 \Rightarrow x_{12} \leftarrow 1$
$I \leftarrow \{6, 7, 8, 9, 10\}$
step 3 : $Limit \leftarrow 2.4 \Rightarrow RCL \leftarrow \{10\}$
$i^* \leftarrow 10 \Rightarrow x_{10} \leftarrow 1$
$I \leftarrow \{6\}$
step 4 : $Limit \leftarrow 6.0 \Rightarrow RCL \leftarrow \{6\}$
$i^* \leftarrow 6 \Rightarrow x_6 \leftarrow 1$
$I \leftarrow \emptyset$

Fig. 5. A second greedy phase for the didactic exemple

B. The second implementation of GRASP for general SPP

The second implementation is inspired by a GRASP implementation for the Node Packing Problem [13], [16]. It is more efficient when the optimal value is small compared to the variable number.

B.1 The greedy randomized phase

The greedy procedure (Algorithm 3) builds a solution from a trivial admissible solution ($x_i = 0, \forall i \in I$). Some variable values are changed (ie fixed to 1), keeping an admissible solution. Changes concern only one variable for one iteration. In order to increase the objective function, variables which concern a minimum number of constraints and with a maximum value are prioritized, but the choice is random among the most interesting variables. Changes stop when we can not fix a variable to 1 without the solution becoming non-admissible.

$x_i \leftarrow 0, \forall i \in I$
$Eval_i \leftarrow \sum_{j \in J} t_{i,j}/c_i, \forall i \in I$
while $(I \neq \emptyset)$ loop
| $Limit \leftarrow (2 - \alpha) * \min_{i \in I}(Eval_i)$
| $RCL \leftarrow \{i \in I, Eval_i \leq Limit\}$
| $i^* \leftarrow RandomSelect(RCL)$
| $x_{i^*} \leftarrow 1$
| $I \leftarrow I \setminus \{i \in I, \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$
endWhile

Alg. 3. The second greedy randomized algorithm

We can also note that the fixation to 1 of a variable does not change the evaluation of variables which can still be fixed to 1 because saturated constraints do not concern them. So, the variable evaluation can be realized once and once only.

An exemple of a possible greedy phase for the didactic exemple with $\alpha = 0.8$ is presented in Figure 5.

B.2 The local search phase

The neighbourhood used for local search procedure is also $k - p$ exchanges. However, tests for $0 - 1$ exchanges are useless after this greedy procedure (no variable can be fixed to 1).

C. The implementation of GRASP for our particular SPP

Due to our problem particularities ($z \leq n$), we decided to use the second implementation of GRASP. However, as our problem is unicost, some exchanges do not have to be tested : $1-1$ exchanges and $2-1$ exchanges can not increase objective function value. So, we only tested $1-2$ exchanges for local search.

IV. Computational Results

In this section, we report computational results obtained for several situations of our real railway problem. We compare the solutions generated with our GRASP implementation and the Cplex solver.

The characteristics of these instances are presented in the Table I. The value of the linear relaxation (LP) is also shown when it is known. In some case, a value is not available (N.A.) due to internal error reported by Cplex. Moreover, the number of non-redundant variables, obtained after application of a dominance test between all pair of variables, is given, as the reduction obtained.

These results are obtained on a Pentium with 800 MHz for GRASP and on a UltraSPARC-II with 296 MHz for Cplex. Our implementation of GRASP has been realized with Ada (Gnat 3.13). We consider these parameters : first the greedy method and neighbourhood described in section III-C, and second the generation of 50 solutions as stopping criterion. The value considered for the alpha parameter has been determined according to results observed in Table II which indicates the best results obtained on each instance for each alpha value (note that one value is not available : due to a very low value obtained with a random initial solution, the local search phase needs too many time).

Some results reported in this table seem amazing compared with the usual tuning reported in the GRASP papers : for some instances, the best results are obtained with low alpha values. Nevertheless, in looking the characteristics (i.e. the maximum, mean and minimum values, see figure 6) and the distribution (see figure 7) of the 50 solutions generated for several alpha value for one of these instances, we can observe that the best solutions are exceptional. If no value can be highlighted as the best on each instance, the value 1.0 gives us the best results on average. This could be explained considering all instances are uni-

N°	Initial problem					Reducted problem	
	Subset	Variables	Constraints	Density	LP	Variables	Reduction
1	16	56	200	5.70%	12.81	26	53.6%
2	16	88	316	5.50%	16.00	64	27.3%
3	16	104	391	5.00%	16.00	73	29.8%
4	18	93	361	4.40%	16.53	40	57.0%
5	20	112	116	4.20%	20.00	20	82.1%
6	24	124	428	3.30%	22.57	80	35.5%
7	50	281	732	1.70%	49.50	164	41.6%
8	90	465	3,025	0.85%	81.04	304	34.6%
9	120	620	4,080	0.64%	108.04	409	34.0%
10	240	1,240	8,300	0.32%	216.04	829	33.1%
11	240	1,240	17,573	0.32%	N.A.	1 099	11.4%
12	360	1,860	19,278	0.21%	313.57	1 482	20.3%
13	360	1,860	39,101	0.22%	N.A.	1 592	14.4%
14	380	1,620	33,664	0.21%	N.A.	1 356	16.3%
15	720	3,720	54,053	0.11%	N.A.	3 339	10.2%
16	720	3,720	158,156	0.11%	590.60	3 307	11.1%
17	2,160	11,160	1,434,968	0.04%	N.A.	10 336	7.4%

TABLE I

Instance characteristics

N°	Rnd	Alpha values												
		0.0	0.15	0.3	0.45	0.5	0.6	0.7	0.75	0.8	0.85	0.9	0.95	1.0
1	9	9	8	9	8	9	9	8	8	8	8	8	8	8
2	16	16	16	16	16	16	16	16	16	16	16	16	16	16
3	16	16	16	16	16	16	16	16	16	16	16	16	15	15
4	12	12	12	12	12	12	12	12	12	12	12	12	12	12
5	20	20	20	20	20	20	20	20	20	20	20	20	20	20
6	20	19	20	20	20	20	20	20	20	20	20	20	20	20
7	46	48	48	48	48	48	48	48	48	48	48	48	48	48
8	39	40	41	41	38	38	38	40	38	39	41	39	**44**	**44**
9	51	51	51	52	50	51	52	53	51	53	54	53	**59**	**59**
10	99	101	101	105	99	99	100	101	101	102	104	106	**119**	**119**
11	80	91	90	89	89	**93**	88	87	89	83	80	80	80	80
12	135	143	145	146	147	**150**	144	145	146	148	145	149	**150**	**150**
13	68	68	71	70	70	71	72	72	**73**	72	72	72	**73**	**73**
14	76	79	78	82	82	83	84	83	85	82	84	84	**86**	85
15	237	**267**	**267**	262	265	263	261	263	261	240	239	240	240	240
16	81	85	82	86	**87**	86	82	84	83	83	83	82	82	84
17	N.A.	87	87	87	88	88	89	**90**	**90**	**90**	88	**90**	89	**90**

TABLE II

Computational results for different values of alpha parameter

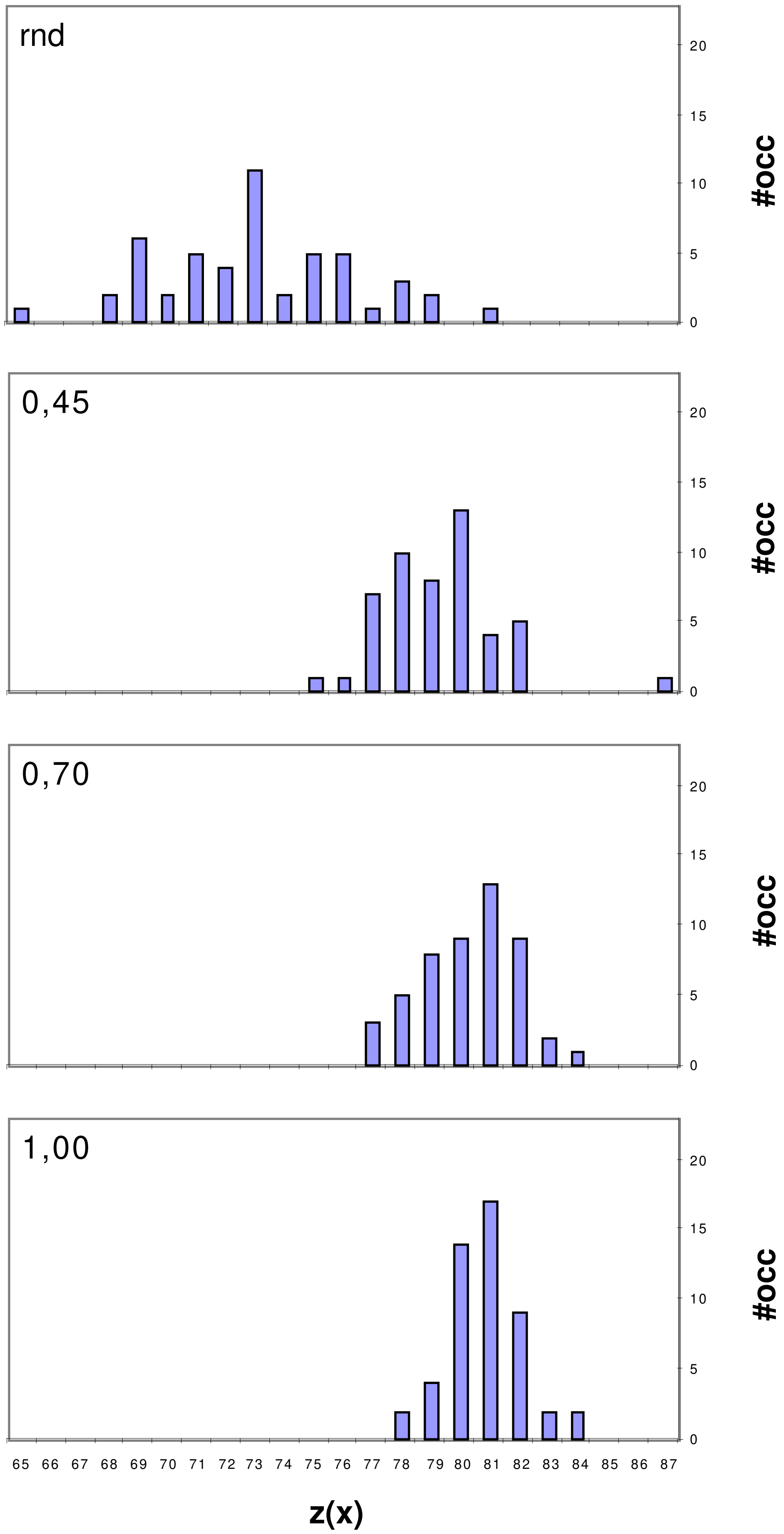

Fig. 7. Distribution of the results for the instance 16

N°	Cplex	Cplex after reduction tests	Best bound	Greedy $\alpha = 1.0$	GRASP $\alpha = 1.0$	GRASP Best value
1	9	9	9.00	8	8	9
2	16	16	16.00	15	16	16
3	16	16	16.00	15	15	16
4	12	12	12.00	12	12	12
5	20	20	20.00	20	20	20
6	20	20	20.00	18	20	20
7	48	48	48.00	48	48	48
8	45	45	45.00	44	44	44
9	60	60	60.00	59	59	59
10	120	120	120.00	119	119	119
11	N.A.	94	94.00	79	80	93
12	149	150	156.04	148	150	150
13	N.A.	N.A.	N.A.	69	73	73
14	N.A.	N.A.	N.A.	82	85	86
15	N.A.	280	283.21	236	240	267
16	69	54	540.11	78	84	87
17	N.A.	N.A.	N.A.	82	90	90

TABLE III
Computational results

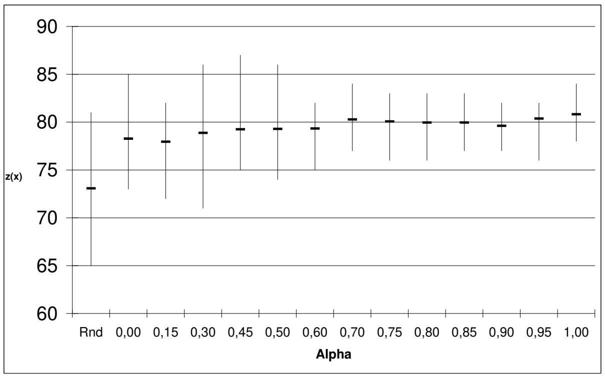

Fig. 6. Computational results for the instance 16

cost and so, many variables have the same evaluation value. So, even if a value of 1.0 for the alpha parameter remove one of the characteristics of a GRASP algorithm, we have considered this value for the comparison with Cplex. You can also note the importance of the greedy phase : results obtained with a random initial solution are much worse.

We used MIP-Solver of Cplex 6.0 to solve exactly our instances when it was possible. When Cplex can not find the optimal solution in reasonable time, the best solution obtained is indicated. Due to the bad quality of the LP relaxation, the use of clique cuts is generally more efficient than the use of Cplex with default parameters [1]. So the best results on the biggest instances (more than 100 variables) were obtained when these cuts are setting on.

The experiments enable us to obtain computational results (Table III) and times (Table IV) in which we can distinguish four main cases :

- small sized instances (N° 1 to 7) :

We can obtain an optimal solution with Cplex in short time. GRASP also obtains good quality results (the value of the objective function is equal to the optimal value except for two instances, optimal value is always found with some alpha values) in equivalent times.

- medium sized instances (N° 8 to 10) :

We can obtain an optimal with Cplex but in an important time. Reduction test becomes very important in order to obtain good results in a reasonable time. GRASP enables to obtain good quality results (less than 2.2% of difference with optimal value) in equivalent times.

- large sized instances (N° 11 to 16) :

We can not obtain optimal value, except for the instance 11 with reduction tests. Even with reduction tests, we can not obtain an admissible solution for some instances with Cplex. We obtain good results on the majority of the instances in shorter times with GRASP. However, we can observe important differences between the value obtained with different value for the alpha parameter.

- very large sized instances (N° 17) :

We can not obtain an admissible solution in a reasonable time with Cplex. GRASP enables to obtain good results but its computational time is very important.

We can also note that if the local search used permits to increase significantly the objective function value obtained for some large sized instances, it takes a more and more important part of computational time when the instance size increases. It seems that local search should be improved in

N°	Cplex	Cplex after reduction tests	Greedy phase	GRASP
1	0.1	0.0	0.0	0.0
2	0.1	0.0	0.1	0.1
3	0.2	0.1	0.0	0.2
4	0.2	0.0	0.1	0.1
5	0.1	0.0	0.1	0.1
6	0.3	0.1	0.1	0.4
7	0.4	0.1	0.6	1
8	130	7	2	5
9	638	19	5	10
10	200 000	45	26	53
11	N.A.	2 265	34	226
12	200 000	200 000	73	344
13	N.A.	N.A.	68	399
14	N.A.	N.A.	57	383
15	N.A.	200 000	304	3 874
16	200 000	200 000	302	4 350
17	N.A.	N.A.	2 776	96 468

TABLE IV
Computational times (in seconds)

order to consider very large sized instances in reasonable time.

V. Conclusion and Perspectives

In this paper, we have considered a real railway problem which can be formulated as a classic difficult combinatorial optimization problem, the Set Packing Problem. We also presented an heuristic algorithm issued from the metaheuristic GRASP to solve this problem. Comparison between this heuristic and a well-known commercial software (Cplex) on different instances enables us to show that GRASP is really more efficient than Cplex for medium and large sized instances.

However, we also observed that even GRASP has some difficulties to solve the biggest instances in a reasonable time. So we are examining different tracks to improve GRASP performances on this problem. According to these experiments, we will be able to adjust GRASP parameters for this type of problem, maybe in considering some values for alpha instead of only one. We will also try to reconsider the local search phase algorithm which represents the main part of the computational time. An improvement could result from a reduction of the research area for $1-2$ exchanges or from a different local search algorithm (tabu search for exemple). In addition, we will study the impact of some other preprocessing phases to reduce the problem size : reduction tests [9] and valid inequalities [7] are our two main tracks. Finally, Node Packing and Set Covering Problem formulation would be studied despite the improvement of the problem size induced.

References

[1] Michel Gondran and Michel Minoux, *Graphes et algorithmes*, Eyrolles, 1995, 588 p. In french.

[2] George L. Nemhauser and Laurence A. Wolsey, *Integer and combinatorial optimization*, Willey-Interscience, 1999, 763 p.

[3] Xavier Delorme, Xavier Gandibleux, and Joaquin Rodriguez, “Application de la métaheuristique GRASP à la résolution d’un problème de capacité d’infrastructure ferroviaire,” Francoro III, May, 9-12 2001, Québec (Canada).

[4] Xavier Delorme, Joaquin Rodriguez, and Xavier Gandibleux, “Heuristics for railway infrastructure saturation,” in *ATMOS 2001 (satellite workshops of the 28th international colloquium on automata, languages, and programming, ICALP) proceedings. Electronic Notes in Theoretical Computer Science.* vol. 50, pp. 41–55, Elsevier Science, URL: http://www.elsevier.nl/locate/entcs/volume50.html.

[5] “Using the CPLEX callable library (manuel), version 4.0, CPLEX optimization, 1995,” .

[6] Thomas A. Féo and Mauricio G.C. Resende, “A probabilistic heuristic for a computationally difficult set covering problem,” *Operations Research Letters*, vol. 8, pp. 67–71, 1989.

[7] Manfred W. Padberg, “On the facial structure of set packing polyhedra,” *Mathematical Programming*, vol. 5, pp. 199–215, 1973, North-Holland Publishing Company.

[8] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco, “An exact algorithm for the project scheduling with ressource constraints based on a new mathematical formulation,” *Management Science*, vol. 44, no. 5, pp. 714–729, may 1998.

[9] Peter J. Zwaneveld, Léo G. Kroon, H. Edwin Romeijn, Marc Salomon, Stéphane Dauzère-Pérès, Stan P.M. Van Hoesel, and Harrie W. Ambergen, “Routing trains through railway stations : Model formulation and algorithms,” *Transportation Science*, vol. 30, no. 3, pp. 181–194, august 1996.

[10] Leonidas S. Pitsoulis and Mauricio G.C. Resende, “Greedy randomized adaptive search procedures,” in *Handbook of Applied Optimization*, P.M. Pardalos and M.G.C. Resende, Eds. Oxford University Press, 2001.

[11] Paola Festa and Mauricio G.C. Resende, “Grasp : an annoted bibliography,” in *Essays and surveys on metaheuristics*, Pierre Hansen and Celsio Ribeiro, Eds. Kluwer academic publishers, 2001.

[12] Xavier Delorme, *Optimisation combinatoire et problèmes de capacité d’infrastructure ferroviaire*, Mémoire de DEA, Université de Valenciennes et du Hainaut Cambrésis, Valenciennes, France, 2000, In french.

[13] Thomas A. Féo and Mauricio G.C. Resende, “Greedy random-

ized adaptative search procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[14] Xavier Gandibleux, David Vancoppenolle, and Daniel Tuyttens, "A first making use of GRASP for solving MOCO problems," 14th International Conference in Multiple Criteria Decision-Making, June, 8-12 1998, Charlottesville, USA.

[15] David Vancoppenolle, *Résolution par GRASP de problèmes d'optimisation combinatoire*, Mémoire de 2ème licence en informatique, Université de Mons-Hainaut, Mons, Belgique, 1998, In french.

[16] Thomas A. Féo, Mauricio G.C. Resende, and Stuart H. Smith, "A greedy randomized adaptative search procedure for maximum independant set," *Operation Research*, vol. 42, pp. 860–878, 1994.