Laboratoire d'Automatique, de Mécanique et d'Informatique industrielles et Humaines — UMR 8530 CNRS

Recherche Opérationnelle et Informatique

Note de Recherche

n° 13 / 2004 / LAMIH-ROI

An Ant Colony Algorithm for the Set Packing Problem

Xavier Gandibleux, Xavier Delorme, Vincent T'Kindt

Submitted March 24, 2004
to Ants'04, fourth International Workshop on Ant Colony Optimization and Swarm Intelligence

UNIVERSITÉ DE VALENCIENNES ET DU HAINAUT-CAMBRÉSIS
LE MONT HOUY — F-59313 VALENCIENNES CEDEX 9 — FRANCE
Secrétariat : Mme Aureggi – phone: +33 (0)3 27 51 19 41
`www.univ-valenciennes.fr/LAMIH`

An Ant Colony Optimisation Algorithm for The Set Packing Problem

Xavier GANDIBLEUX[1], Xavier DELORME[1], and Vincent T'KINDT[2]

[1] LAMIH/ROI – UMR CNRS 8530
Université de Valenciennes, Campus "Le Mont Houy"
F-59313 Valenciennes cedex 9 - FRANCE
{Xavier.Gandibleux,Xavier.Delorme}@univ-valenciennes.fr

[2] Laboratoire d'Informatique
Polytech'Tours
64 avenue Jean Portalis
F-37200 Tours - FRANCE
Tkindt@univ-tours.fr

Abstract. In this paper we consider the application of an Ant Colony Optimisation (ACO) metaheuristic on the Set Packing Problem (SPP) which is a NP-hard optimisation problem. For the proposed algorithm, two solution construction strategies based on exploration and exploitation of solution space are designed. The main difference between both strategies concerns the use of pheromones during the solution construction. The selection of one strategy is driven automatically by the search process. A territory disturbance strategy is integrated in the algorithm and is triggered when the convergence of the ACO stagnates. A set of randomly generated numerical instances, involving from 100 to 1000 variables and 100 to 5000 constraints, was used to perform computational experiments. To the best of our knowledge, only one other metaheuristic (Greedy Randomized Adaptative Search Procedure, GRASP) has been previously applied to the SPP. Consequently, we report and discuss the effectiveness of ACO when compared to the best known solutions and including those provided by GRASP. Optimal solutions obtained with Cplex on the smaller instances (up to 200 variables) are indicated with the calculation times. These experiments show that our ACO heuristic outperforms the GRASP heuristic. It is remarkable that the ACO heuristic is made up of simple search techniques whilst the considered GRASP heuristic is more evolved.

1 Introduction

The set packing problem (SPP) is formulated as follows. Given a finite set $I = \{1, \ldots, n\}$ of items and $\{T_j\}, j \in J = \{1, \ldots, m\}$, a collection of m subsets of I, a packing is a subset $P \subseteq I$ such that $|T_j \cap P| \leq 1, \forall j \in J$. The set J can be also seen as a set of exclusive constraints between some items of I. Each item $i \in I$ has a positive weight denoted by c_i and the aim of the SPP is to calculate the packing which maximises the total weight. This problem can be formulated by integer programming as follows:

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

In the above model, the variables are the x_i's with $x_i = 1$ if item $i \in P$, and $x_i = 0$ otherwise. The data $t_{i,j}, \forall i \in I, \forall j \in J$, enable us to model the exclusive constraints with $t_{i,j} = 1$ if item i belongs to set T_j, and $t_{i,j} = 0$ otherwise.
Notice that the special case in which $\sum_{i \in I} t_{i,j} = 2, \forall j \in J$, is the node packing problem.

The SPP is known to be strongly NP-Hard, according to Garey and Johnson [6]. The most efficient exact method known for solving this problem (as suggested in [9]) is a Branch & Cut algorithm based on polyhedral theory and the works of Padberg [12] to obtain facets. However, only small-sized instances can be solved to optimality. To the best of our knowledge, and also according to Osman and Laporte [10], few metaheuristics have been applied to the solution of the SPP. Besides, few applications have been reported in the literature. Rönnqvist [13] worked on a cutting stock problem formulated as a SPP and solved it using a lagrangian relaxation combined with subgradient optimisation. Kim [7] modelled a ship scheduling problem as a SPP and used LINDO software to solve it. Mingozzi et al. [8] used a SPP formulation to calculate bounds for a resource constrained project scheduling problem. This SPP formulation is solved by a greedy algorithm. At last, Rossi [14] modelled a ground holding problem as a SPP and solved it with a Branch & Cut method.
The application of the SPP to a railway planning problem has been first studied by Zwaneveld et al. [18]. Railway infrastructure managers now have to deal with operators' requests for increased capacity. Planning the construction or reconstruction of infrastructures must be done very carefully due to the huge required investments. Usually, assessing the capacity of one component of a rail system is done by measuring the maximum number of trains that can be operated on this component within a certain time period. Measuring the capacity of junctions is a matter of solving an optimisation problem called the *feasibility problem*, and which can be formulated as a SPP. Zwaneveld et al. proposed reduction tests and a Branch & Cut method to solved it to optimality.
More recently, Delorme et al. have taken up again this application of the SPP and proposed heuristic algorithms [1, 3]. Among these heuristics, the most efficient one is derived from the metaheuristic GRASP [2]. The designed algorithm integrates advanced strategies like a path-relinking technique, a learning process and a dynamic tuning of parameters. Besides, it should be noticed that basi-

cally GRASP is not a *population based* heuristic since at each iteration of the algorithm only one solution is considered.

In this paper, we consider an implementation of Ant Colony Optimisation (ACO) principles on the SPP. ACO algorithms are population based heuristics in which, at each iteration, ants build solutions by exploiting a common memory. Henceforth, ACO algorithms have the capability of learning “good and bad” decisions when building solutions. This motivated the design of the proposed ACO heuristic for the SPP, in which taking a decision consists in choosing if a given item belongs to a packing under construction. Since the last decade, ACO algorithms become more and more used in the field of Operations Research ([4]). Notably, they have been applied to the solution of the quadratic assignment problem ([5]) and the traveling salesman problem ([16]).
The remainder of the paper is organised as follows. Section 2 is devoted to the presentation of the designed ACO heuristic and its implementation. Section 3 deals with numericals experiments conducted on various kind of instances. Our ACO heuristic is compared to the GRASP heuristic presented in [1, 2] and to Cplex solver applied on Model 1. We conclude this section and the paper by providing an analysis of the behavior of our ACO heuristic.

2 The Ant Colony Optimisation algorithm for the SPP

The basic idea of ACO algorithms comes from the capability of ants to find shortest paths from their nest to food locations. For combinatorial optimization problems this means that ants search how to build good solutions. At each iteration of this search, each ant builds a solution by applying a constructive procedure which uses the common memory of the colony. This memory, referred to as the *pheromone matrix*, corresponds to the pheromone trails for real ants. Rougthly speaking, the pheromone matrix contains for a combinatorial optimisation problem, the probabilities of building good solutions. Once a complete solution has been computed, pheromone trails are updated according to the quality of the best solution built. Hence, cooperation between ants is performed by exploiting the pheromone matrix. In this paper we use some principles of the ACO framework, denoted by SACO, proposed by Stuztle [15] and revised successfuly by T'kindt et al. [17] for a scheduling problem. Basically, when constructing a solution, an ant iteratively takes decisions either in the *exploration mode* or in the *exploitation mode*. In a basic ACO framework the choice of a mode is done following a fixed known probability. However, in the SACO framework we use an important feature of Simulated Annealing algorithms which is to allow more diversification at the beginning of the solution process and more intensification at the end. Henceforth, in the SACO framework the above mentionned probability evolved along the solution process. Besides, we also consider that after having built a solution an ant applies on it a local search. A territory disturance strategy is integrated in this framework. This strategy is inspired from a warming up strategy, well-known for the simulated simulated annealing metaheuristic. We now describe more accurately the ACO heuristic designed for the SPP.

Algorithm 1 The main procedure

```
--| Generate an initial solution using a greedy algorithm and a local search
elaborateSolutionGreedy( sol ↑ )
localSearch( sol ↕ )
copySolution( sol ↓ , bestSolKnown ↑ )

--| ACO Algorithm
initPheromones( φ ↑ ); iter ← 0
while not( isFinished?( iter ↓ ) ) do
  resetToZero( bestSolIter ↑ )
  for ant in 1... maxAnt do
    if isExploitation?(ant ↓, iter ↓, iterOnExploit ↓, maxIter ↓ ) then
      elaborateSolutionGreedyPhi( φ ↓ , solution ↑ )
    else
      elaborateSolutionSelectionMethod( φ ↓ , solution ↑ )
    end if
    localSearch( sol ↕ )
    if performance( sol ) > performance( bestSolIter ) then
      copySolution( sol ↓ , bestSolIter ↑ )
      if performance( sol ) > performance( bestSolKnown ) then
        copySolution( sol ↓ , bestSolKnown ↑ )
      end if
    end if
  end for
  managePheromones( φ ↕ , bestSolKnown ↓ , bestSolIter ↓ )
  iter++
end while

--| The best solution found
putLine( bestSolKnown ↓ )
```

The general outline of the proposed ACO heuristic is given in Algorithm 1. Initially, a greedy heuristic is applied to provide the initial best known solution. It works as follows (procedure `elaborateSolutionGreedy`, see Algorithm 2). Iteratively the candidate variable which involves a minimum number of constraints with a maximum value is selected. This process is repeated until there is no more candidate variable available.

Algorithm 2 The `elaborateSolutionGreedy` procedure

$I_t \leftarrow I \;\; ; \quad x_i \leftarrow 0, \forall i \in I_t$
$valuation_i \leftarrow c_i / \sum_{j \in J} t_{i,j}, \forall i \in I_t$
while $(I_t \neq \emptyset)$ **do**
  $i^* \leftarrow \text{bestValue}(valuation_i, i \in I_t)$
  $x_{i^*} \leftarrow 1 \;\; ; \quad I_t \leftarrow I_t \setminus \{i^*\} \;\; ; \quad I_t \leftarrow I_t \setminus \{i : \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$
end while

The neighbourhood $\mathcal{N}$ used for the local search procedure is based on $k-p$ exchanges. The $k-p$ exchange neighbourhood of a solution x is the set of solutions obtained from x by changing the value of k variables from 1 to 0, and changing p variables from 0 to 1. Due to the combinatorial explosion of the number of possible exchanges when k and p increase, we decided to implement the $1-1$ exchanges. This exchange is valuable only for weighted instances, *i.e.* instances in which there exists, at least, c_i and c_j such that $c_j \neq c_i$. Consequently, no local search is applied for unicost instances, *i.e.* those instances with $c_i = c_j$, $\forall i, j \in I$. Moreover, the search procedure was implemented using a non-iterative first-improving strategy (*i.e.* we selected the first neighbour whose value is better than the current solution).

Algorithm 3 The `elaborateSolutionSelectionMode` procedure

$I_t \leftarrow I$; $x_i \leftarrow 0, \forall i \in I_t$
$\mathcal{P} \leftarrow \log_{10}(\texttt{iter}) / \log_{10}(\texttt{maxIter})$
while $(I_t \neq \emptyset)$ **do**
  if (randomValue(0,1) $> \mathcal{P}$) **then**
    $i^* \leftarrow$ rouletteWheel$(\phi_i, i \in I_t)$
  else
    $i^* \leftarrow$ bestValue$(\phi_i, i \in I_t)$
  end if
  $x_{i^*} \leftarrow 1$; $I_t \leftarrow I_t \setminus \{i^*\}$; $I_t \leftarrow I_t \setminus \{i : \exists j \in J, t_{i,j} + t_{i^*,j} > 1\}$
end while

Let ϕ be the pheromone matrix and ϕ_i be the probability of having item i in a good packing for the SPP. Initially, the pheromones are initialized (routine `initPheromones`) by assigning $\phi_i \leftarrow$ `phiInit` for all variables $i \in I$, with `phiInit` a given parameter. Each ant elaborates a feasible saturated solution (*i.e.* a solution in which it is impossible to add one more variable without violating the constraints set) starting from the trivial feasible solution, $x_i = 0, \forall i \in I$. Some variable are set to 1, as long as the solution is maintained feasible. Changes concern only one variable at each step and there is no more change when no variable can be fixed to 1 without losing feasibility. The choice of a variable x_i to be set to 1 is done either in the *exploration mode* or the *exploitation mode* according to the procedure `elaborateSolutionSelectionMode` described in Algorithm 3. In the exploration mode a roulette wheel is applied on the set of candidate variables whilst in the exploitation mode the candidate variable with the greatest value of pheromone is selected. The ceil probability $\mathcal{P}$ evolves along the solution process following a logarithmic curve regularly restarted. This mechanism enables the ants to periodically strongly diversify their search for a good solution. Notice that for some ants when the predicate `isExploitation?` is true, a solution is built by applying the greedy strategy on the current level of pheromones (like in procedure `elaborateSolutionGreedy` but with $valuation_i = \phi_i$). The above predicate is true for each first ant of an iteration, every `iterOnExploit` iterations.

After all ants have built a solution the best one for the current iteration is retained and *evaporation* and *deposition* of pheromones is performed. It means that we increase the pheromones ϕ_i corresponding to the items selected in the

Algorithm 4 The `managePheromones` procedure

```
--| Pheromone evaporation
for i in 1 . . . ncol do
   φ_i ← φ_i ∗ rhoE
end for

--| Pheromone deposition
for i in 1. . . ncol do
   if (bestSolIter.x[i] = 1) then
      φ_i ← φ_i + rhoD
   end if
end for

--| Territory disturbance
if      isStagnant?( bestSolution , iterStagnant )
   and isExistsPhiNul?( φ )
   and isRestartEnable?( iter , lastIterRestart , maxIteration ) then
   --| Disturb the pheromones
   for i in 1. . . ncol do
      φ_i ← φ_i ∗ 0.95 ∗ log10(iter)/log10(maxIteration)
   end for
   for i in 1. . . random(0.0, 0.1 ∗ ncol) do
      φ_random(1,ncol) ← random(0.05, (1.0 − iter/maxIteration) ∗ 0.5)
   end for
   --| Offset on the pheromones with low level
   for i in 1. . . ncol do
      if φ_i < 0.1 then
         φ_i ← φ_i + random(0.05, (1.0 − iter/maxIteration) ∗ 0.5)
      end if
   end for
end if
```

best packing of the current iteration, whilst we decrease the other pheromones. This process is described in the procedure `managePheromones` (Algorithm 4). Besides a disturbance strategy has been integrated to this management procedure. This strategy is triggered when three conditions are true: (1) the convergence of the ACO is in stagnation (predicate `isStagnant?` is true), (2) at least one pheromone has its level set to zero (predicate `isExistsPhiNul?` is true), and (3) it remains enough iterations after the application of the disturbance to stabilize the pheromones (predicate `isRestartEnable?` is true). Finally, the procedure is stopped after a predefined number of iterations (predicate `isFinished?` is true). The parameters used in the ACO heuristic are reported in Table 1.

`maxIter`	200	Number of iterations determined a priori
`maxAnt`	15	Number of ants for each iteration
`phiInit`	1.0	Initial pheromone assigned to a variable
`rhoE`	0.8	Pheromone evaporation rate
`rhoD`	phiInit * (1.0 - rhoE)	Pheromone deposition rate
`phiNul`	0.001	Level of pheromon considered as zero
`iterOnExploit`	0.750	Percentage of iterations when Exploitation mode is activated
`iterStagnant`	8	Declare the procedure stagnant when no improvement is observed

Table 1. The parameters

3 Numerical Experiments and Analysis

This section presents the computational results obtained on all the randomly generated instances considered in our study. The solutions generated by GRASP and our ACO implementation are included. Our implementation of ACO has been performed with C language whilst GRASP has been developped with Ada Language. The results were obtained on a PC Pentium III at 800 MHz for GRASP and ACO. We also compare these heuristics to the optimal solutions calculated by Cplex solver for instances with up to 200 variables. As Cplex was not capable of solving all larger instances, we consider in this case the best known integer solution which is compared to the two heuristics. Roughly speaking, this best solution is taken, for a given instance, as the one returned by ACO, GRASP and Cplex (when time limited) which yields the highest value of the total cost.
Characteristics of the instances and results provided by Cplex and the heuristics are given in Tables 2 and 3. In each of these tables and for each instance, the column *#var* contains the number of variables and the column *#cst* contains the number of constraints. The *Density* column corresponds to the percentage of non-null elements in the constraint matrix. The two remaining columns describing the characteristics of the instances are $MaxOne$ which provides the maximum number of data $t_{i,j}$ different from 0, and $Weight$ which indicates the interval in which the costs c_i are comprised. Notice that instances for which the interval is $[1-1]$ are instances of the unicost SPP. For all the heuristics we report the average objective function value (column *Avgvalue*) found over 16 runs. We also give the maximum objective function value found (column *Bestvalue*) and the average required CPU time (column $CPUt$).
All the tested instances are freely available on the Internet at the address `www3.inrets.fr/~delorme/Instances-fr.html`.

We first focus on the behavior of the ACO heuristic on the instance pb500rnd15. Figure 1 reports the evolution of the best solution calculated (*BestGlobal* curve) as well as the best solution of each iteration (*BestIteration*). It appears that the evolution of the ceil probability $\mathcal{P}$ implies a quick convergence towards a good global solution value. However, the best solution is obtained thanks to the disturbance strategy. This is a typical behavior of the designed ACO.

Table 2 presents the results for instances with up to 200 variables. Cplex solver was capable of solving all these instances to optimality. Henceforth, column $0/1$*solution* reports, for each instance, the information about the optimal solution value as well as the CPU time required by Cplex to calculate it. Notice that on some instances this CPU time is prohibitive.
It appears that the ACO found in the best case the optimal solutions except for one instance. Besides, the best solution found by ACO always outperforms the best solution value obtained by GRASP. The comparison of average calculated solution values of ACO and GRASP show that on instances with 100 variables GRASP slightly outperforms ACO, and only on unicost instances. On instances with 200 variables ACO strictly outperforms GRASP 8 times whilst

Table 2: Instances and results (1/2)

Instances	Characteristics					0/1 Solution		GRASP			ACO			
	#var	#cst	Density	Max One	Weight	Optimal value	CPUt (s)	Best value	Avg value	CPUt (s)	Best value	Avg value	CPUt (s)	Optimal found
pb100rnd01	100	500	2.0%	2	[1-20]	372	2.92	372	372.00	1.97	372	372.00	3.33	•
pb100rnd02	100	500	2.0%	2	[1-1]	34	0.60	34	34.00	1.31	34	34.00	2.00	•
pb100rnd03	100	500	3.0%	4	[1-20]	203	7.81	203	203.00	1.14	203	203.00	2.00	•
pb100rnd04	100	500	3.0%	4	[1-1]	16	52.86	16	16.00	1.29	16	15.56	0.67	•
pb100rnd05	100	100	2.0%	2	[1-20]	639	0.01	639	639.00	0.80	639	639.00	1.67	•
pb100rnd06	100	100	2.0%	2	[1-1]	64	0.01	64	64.00	0.69	64	64.00	1.00	•
pb100rnd07	100	100	2.9%	4	[1-20]	503	0.00	503	503.00	1.00	503	503.00	1.00	•
pb100rnd08	100	100	3.1%	4	[1-1]	39	0.02	39	38.75	0.57	39	38.68	0.67	•
pb100rnd09	100	300	2.0%	2	[1-20]	463	0.49	463	463.00	1.26	463	463.00	1.67	•
pb100rnd10	100	300	2.0%	2	[1-1]	40	1.13	40	40.00	1.28	40	39.62	1.00	•
pb100rnd11	100	300	3.1%	4	[1-20]	306	0.48	306	306.00	0.68	306	306.00	1.67	•
pb100rnd12	100	300	3.0%	4	[1-1]	23	6.80	23	23.00	1.13	23	22.93	0.33	•
pb200rnd01	200	1000	1.5%	4	[1-20]	416	8 760.73	416	415.18	7.32	416	415.25	27.33	•
pb200rnd02	200	1000	1.5%	4	[1-1]	32	156 109.36	32	32.00	7.35	32	31.56	14.67	•
pb200rnd03	200	1000	1.0%	2	[1-20]	731	5 403.23	726	722.81	10.81	729	725.12	44.33	○
pb200rnd04	200	1000	1.0%	2	[1-1]	64	63 970.91	63	63.00	9.12	64	62.93	24.33	•
pb200rnd05	200	1000	2.5%	8	[1-20]	184	1 211.37	184	184.00	4.62	184	182.56	16.00	•
pb200rnd06	200	1000	2.5%	8	[1-1]	14	8 068.20	14	13.37	3.48	14	12.87	4.00	•
pb200rnd07	200	200	1.5%	4	[1-20]	1 004	0.02	1002	1001.12	4.20	1004	1003.50	6.33	•
pb200rnd08	200	200	1.5%	4	[1-1]	83	0.04	83	82.87	2.71	83	82.75	2.67	•
pb200rnd09	200	200	1.0%	2	[1-20]	1 324	0.01	1324	1324.00	3.75	1324	1324.00	7.33	•
pb200rnd10	200	200	1.0%	2	[1-1]	118	0.02	118	118.00	3.64	118	118.00	4.00	•
pb200rnd11	200	200	2.5%	8	[1-20]	545	0.33	545	544.75	2.36	545	545.00	4.33	•
pb200rnd12	200	200	2.6%	8	[1-1]	43	1.70	43	43.00	1.01	43	43.00	1.33	•
pb200rnd13	200	600	1.5%	4	[1-20]	571	830.39	571	566.43	6.01	571	568.50	20.33	•
pb200rnd14	200	600	1.5%	4	[1-1]	45	10 066.91	45	45.00	3.92	45	44.43	8.67	•
pb200rnd15	200	600	1.0%	2	[1-20]	926	12.20	926	926.00	4.22	926	926.00	27.00	•
pb200rnd16	200	600	1.0%	2	[1-1]	79	14 372.85	79	78.31	6.80	79	78.37	15.33	•
pb200rnd17	200	600	2.5%	8	[1-20]	255	741.52	255	251.31	3.61	255	253.25	11.00	•
pb200rnd18	200	600	2.6%	8	[1-1]	19	19 285.06	19	18.06	2.35	19	18.12	3.00	•

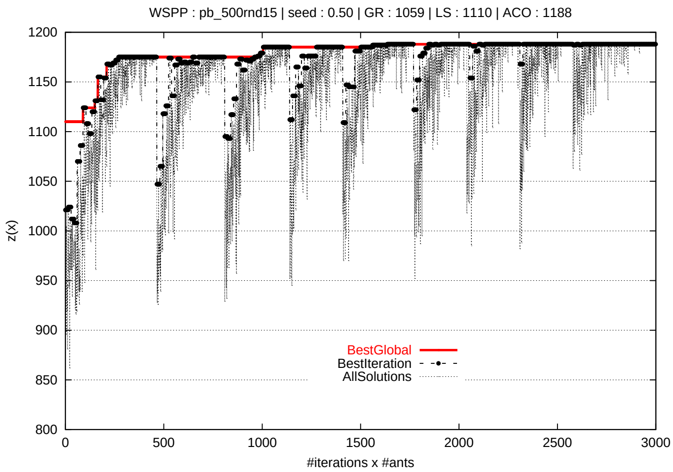

Fig. 1. Behavior of the ACO heuristic on the instance pb500rnd15

the opposite situation occurs 6 times. On this problem size, the comparison of the average values shows that ACO generally gives better results than GRASP. Besides, ACO is dominated by GRASP on unicost instances whilst on weighted instances the converse occurs.
The fact that on unicost instances ACO is slightly outperformed by GRASP is due to the lack of a neighbourhood search applied on the solution calculated by each ant.

Table 3 presents the results for instances with 500 and 1000 variables. As Cplex solver was not capable of solving to optimality all the instances, we report in column $0/1 solution$ the best known solution. When this solution was not shown optimal by Cplex solver, the best solution value indicated is marked by an asterix.
As in Table 2, ACO often found in the best case the best known solution (except for three instances). Besides, it always dominates the best solution value calculated by GRASP, except on the instance pb1000rnd400. The average value obtained by ACO on the unicost instances is always lower than the one of GRASP, except for two instances on which ACO slightly performs better than GRASP. On the weighted instances, GRASP strictly outperforms ACO 6 times whilst the opposite situation also occurs 6 times. From this viewpoint, the two algorithms seem to be complementary. However, it should be noticed that when GRASP provides better averaged results than ACO, it is always of at most 3 units. But when ACO improves over GRASP, the gap between the two averaged values can be up to 25 units.

Figure 2 provides a graphical overview of the performances of the ACO heuristic on all the instances. This figure shows that ACO generally calculates

Table 3: Instances and results (2/2)

Instances	Characteristics					0/1 Solution	GRASP			ACO			
	#var	#cst	Density	Max One	Weight	Best known value	Best value	Avg value	CPUt (s)	Best value	Avg value	CPUt (s)	Best known found
pb500rnd01	500	2500	1.2%	10	[1-20]	323*	323	319.38	32.08	323	319.87	154.67	•
pb500rnd02	500	2500	1.2%	10	[1-1]	24*	24	23.69	25.62	24	23.06	26.67	•
pb500rnd03	500	2500	0.7%	5	[1-20]	776*	772	767.63	70.33	776	772.75	244.00	•
pb500rnd04	500	2500	0.7%	5	[1-1]	61*	61	60.13	57.30	61	60.06	69.00	•
pb500rnd05	500	2500	2.2%	20	[1-20]	122	122	121.50	15.48	122	120.62	71.00	•
pb500rnd06	500	2500	2.2%	20	[1-1]	8*	8	8.00	12.08	8	7.87	9.67	•
pb500rnd07	500	500	1.2%	10	[1-20]	1141	1141	1141.00	13.43	1141	1141.00	60.00	•
pb500rnd08	500	500	1.2%	10	[1-1]	89	89	88.25	15.80	89	88.12	21.67	•
pb500rnd09	500	500	0.7%	5	[1-20]	2236	2235	2235.00	23.44	2236	2234.43	84.33	•
pb500rnd10	500	500	0.7%	5	[1-1]	179	179	178.06	18.20	179	178.31	44.67	•
pb500rnd11	500	500	2.3%	20	[1-20]	424	423	419.31	19.25	424	418.25	37.33	•
pb500rnd12	500	500	2.2%	20	[1-1]	33*	33	33.00	11.91	33	32.62	8.00	•
pb500rnd13	500	1500	1.2%	10	[1-20]	474*	474	470.00	32.88	474	468.00	105.00	•
pb500rnd14	500	1500	1.2%	10	[1-1]	37*	37	36.94	20.77	37	36.50	25.66	•
pb500rnd15	500	1500	0.7%	5	[1-20]	1196*	1196	1186.94	59.36	1196	1190.93	161.67	•
pb500rnd16	500	1500	0.7%	5	[1-1]	88*	88	86.63	36.31	88	86.12	60.67	•
pb500rnd17	500	1500	2.2%	20	[1-20]	192*	192	191.75	18.38	192	188.31	57.00	•
pb500rnd18	500	1500	2.2%	20	[1-1]	13*	13	13.00	12.03	13	12.43	9.00	•
pb1000rnd100	1000	5000	2.60%	50	[1-20]	67	67	65.50	53.50	67	64.00	117.67	•
pb1000rnd200	1000	5000	2.59%	50	[1-1]	4	4	3.15	39.30	4	3.81	15.00	•
pb1000rnd300	1000	5000	0.60%	10	[1-20]	661*	649	639.50	221.20	661	640.50	700.67	•
pb1000rnd400	1000	5000	0.60%	10	[1-1]	48*	48	46.83	149.70	47	45.06	108.33	◦
pb1000rnd500	1000	1000	2.60%	50	[1-20]	222*	222	217.98	64.80	222	219.62	85.67	•
pb1000rnd600	1000	1000	2.65%	50	[1-1]	15*	14	13.68	41.40	15	13.37	8.00	•
pb1000rnd700	1000	1000	0.58%	10	[1-20]	2260	2222	2214.10	119.70	2248	2239.56	296.67	◦
pb1000rnd800	1000	1000	0.60%	10	[1-1]	175*	172	170.81	82.60	173	170.50	94.00	◦

* The asterisks indicate that we don't know if the best known solution is optimal

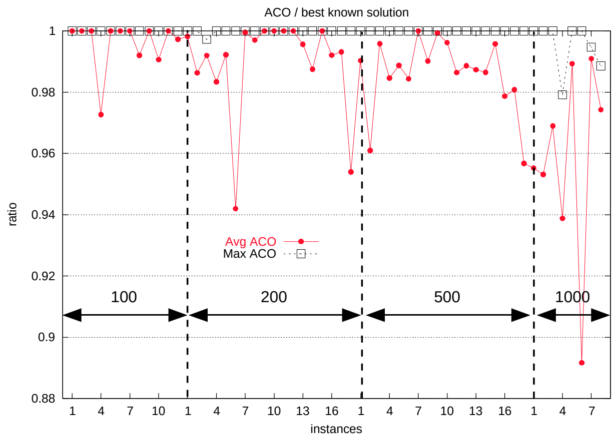

Fig. 2. Comparison between ACO and the best known solution for all instances

good solutions even if, as expected, the gap between the average solution value and the best solution value increases as the problem size increases.
The first conclusion of these experiments is that ACO is capable of finding the best known solutions and providing, at least, similar results than those given by GRASP heuristic. This is an interesting conclusion since the proposed ACO heuristic has a simple structure and does not use evolved search mechanisms as, for instance, the path-relinking process used in the GRASP heuristic. Henceforth, ACO is simpler than GRASP but provides similar or better results. A drawback of ACO is related to the numerical "instability" of the output. In fact, on some instances, when performing 16 runs of this heuristic on the same instances it appears that the 16 returned solution values can be quite distinct. This leads us to the conclusion that more iterations should be allowed to the ACO heuristic to have a stronger convergence and increase the average calculated solution value. But, this would increase the required CPU time.
Besides, from the experiments we can derive that a local search should be applied in the ACO heuristic on unicost instances to provide as good results as on the weighted instances.

The conducted experiments presented in this section clearly show that the ACO heuristic proposed to solve the SPP performs well on the tested instances. Incorporating evolved search mechanisms, as such used in the GRASP heuristic with which we compared, may lead to a highly efficient ACO heuristic.

Acknowledgment. The authors would like to thank Coralie Paccou who worked during her Master thesis in 2001-2002 [11] on a first version of the ACO heuristic, under the supervision of Xavier Gandibleux and Xavier Delorme.

References

1. X. Delorme. *Modélisation et résolution de problèmes liés à l'exploitation d'infrastructures ferroviaires.* PhD thesis, Université de Valenciennes, Valenciennes, France, 2003.
2. X. Delorme, X. Gandibleux, and J. Rodriguez. GRASP for set packing problems. *European Journal of Operational Research*, 153 (3):564–580, 2004.
3. X. Delorme, J. Rodriguez, and X. Gandibleux. Heuristics for railway infrastructure saturation. In L. Baresi, J-J. Lvy, R. Mayr, M. Pezz, G. Taentzer, and C. Zaroliagis, editors, *ICALP 2001*, volume 50 of *Electronic Notes in Theoretical Computer Science (URL: http://www.elsevier.nl/locate/entcs/volume50.html)*, pages 41–55. Elsevier Science, 2001.
4. M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(3):137–172, 1999.
5. L.M. Gambardella, E. Taillard, and M. Dorigo. Ants colonies for the QAP. *Journal of the Operational Research Society*, 5:167–176, 1999.
6. M.R. Garey and D.S. Johnson. *Computers and intractability : a guide to the theory of NP-Completeness.* V.H. Freeman and Company, San Francisco, 1979.
7. S.-H. Kim and K.-K. Lee. An optimization-based decision support system for ship scheduling. *Computers and Industrial Engineering*, 33:689–692, 1997.
8. A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the project scheduling with ressource constraints based on a new mathematical formulation. *Management Science*, 44(5):714–729, mai 1998.
9. G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization.* Willey-Interscience, New York, 1999.
10. I.H. Osman and G. Laporte. Metaheuristics : a bibliography. *Annals of Operations Research*, 63:513–623, 1996.
11. C. Paccou. L'algorithme des fourmis appliqué au Set Packing Problem. Master's thesis, Université de Valenciennes, Valenciennes, France, April 2002.
12. M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.
13. M. Rönnqvist. A method for the cutting stock problem with different qualities. *European Journal of Operational Research*, 83:57–68, 1995.
14. F. Rossi and S. Smriglio. A set packing model for the ground holding problem in congested networks. *European Journal of Operational Research*, 131:400–416, 2001.
15. T. Stuztle. An ant approach for the flow shop problem. In *Proceedings of EU-FIT'98, Aachen (Germany)*, pages 1560–1564, 1998.
16. T. Stuztle and M. Dorigo. ACO algorithms for the traveling salesman problem. *in K. Miettinen, M. Makela, P. Neittaanmaki and J. Periaux (Eds.): Evolutionary Algorithms in Engineering and Computer Science: recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming Genetic Programming and Industrial Applications, John Wiley & Sons*, pages 1560–1564, 1999.
17. V. T'kindt, N. Monmarché, F. Tercinet, and D. Laugt. An ant colony optimization algorithm to solve a 2-machine bicriteria flowshop scheduling problem. *European Journal of Operational Research*, 142(2):250–257, 2002.
18. P.J. Zwaneveld, L.G. Kroon, H.E. Romeijn, M. Salomon, S. Dauzère-Pérès, Stan P.M. Van Hoesel, and H.W. Ambergen. Routing trains through railway stations : Model formulation and algorithms. *Transportation Science*, 30(3):181–194, august 1996.