# An ant colony optimization inspired algorithm for the Set Packing Problem with application to railway infrastructure

Xavier Gandibleux* Julien Jorge† Sébastien Angibaud† Xavier Delorme‡
Joaquin Rodriguez§

*Laboratoire d'Informatique de Nantes Atlantique, Université de Nantes
2 rue de la Houssinière BP92208, F-44322 Nantes cedex 03, France
Xavier.Gandibleux@univ-nantes.fr

†Université de Nantes
2 rue de la Houssinière BP92208, F-44322 Nantes cedex 03, France
{jorge,angibaus}@ensinfo.univ-nantes.fr

‡Centre Génie Industriel et Informatique, École des Mines de Saint-Etienne
158 cours Fauriel, 42023 Saint-Etienne cedex 2, France
delorme@emse.fr

§Institut National de Recherche sur les Transports et leur Sécurité - ESTAS
20 rue Élisée Reclus, F-59650 Villeneuve d'Ascq, France
joaquin.rodriguez@inrets.fr

## 1 Introduction

The paper concerns an Ant Colony Optimisation (ACO) procedure as approximation method for the railway infrastructure capacity (RIC) problem. 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 and the long term implications. 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. In our work, we deal with two real situations. The first is Pierrefitte-Gonnesse crossing point located at the north of Paris. The second is the Lille-Flandres station which is the largest station in North of France. Measuring the capacity of junctions is a matter of solving an optimisation problem called the *saturation problem* [1], and which can be formulated as a Set Packing Problem (SPP). 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 summarized as follows:

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

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.

The SPP is known to be strongly NP-Hard, according to Garey and Johnson [4]. The most efficient exact method known for solving this problem (as suggested in [5]) is a Branch & Cut algorithm based on polyhedral theory and the works initiated by Padberg [7] to obtain facets. Zwaneveld et al. [8] proposed reduction tests and a Branch & Cut method to solve to optimality a railway planning problem written as an SPP. However, only small-sized instances or characteristic configurations of trains can be solved to optimality. To the best of our knowledge, and also according to Osman and Laporte [6], few metaheuristics have been applied to the solution of the SPP. Delorme, Gandibleux and Rodriguez have proposed heuristic algorithms for SPP and evaluated on RIC problems [1, 2]. Recently, Gandibleux, Delorme and T'kindt presented an ACO algorithm for the SPP [3].

In this paper, an evolution of this ACO algorithm is presented. It integrates all features for dealing efficiently with weighted (W-) instances of SPP (*i.e.* instances in which there exists, at least, $c_i$ and $c_j$ such that $c_j \neq c_i$) and unicost (U-) SPP (*i.e.* those instances with $c_i = c_j$, $\forall i, j \in I$). The local search routines have been re-designed and a dynamic stopping condition is integrated. Default value of parameters is integrated in the procedure and used at the startup, avoiding additional efforts for tuning the parameters. The static management of the data structures has been replaced by a dynamic management based on AVL-tree structure. The algorithm is evaluated on RIC instances which are large size USPP problems, and also on a set of public WSPP and USPP instances randomly generated. The results obtained are compared with the former version of our ACO procedure [3] and the best known solutions (the exact solutions when they are available, or approximations obtained with the GRASP [2] and ACO procedures [3]).

## 2 Outline of the ACO heuristic for the SPP

The general outline of the proposed ACO heuristic is given in Algorithm 1. (The arrows $\downarrow$, $\uparrow$ and $\updownarrow$ specify the transmission mode of a parameter to a procedure; they correspond respectively to the mode IN, OUT and INOUT.) Initially, a greedy heuristic is applied to provide the initial best solution. It works as follows (procedure `elaborateSolutionGreedy`). 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

**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 ↑ ); localSearch( sol ↕ )
    else
      elaborateSolutionSelectionMethod( φ ↓ , solution ↑ ); localSearch( sol ↕ )
    end if
    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
```

available. In addition, a local search procedure is applied to this solution. The neighbourhood $\mathcal{N}$ used for this local search procedure is based on a classic $k-p$ 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 $1-2$ exchanges and the $1-1$ exchanges (which are respectively triggered for USPP and WSPP). 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).

Let $\phi$ be the pheromone matrix and $\phi_i$ 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 value. 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*. 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 $\mathcal{P}$ which is used to determine the mode selected, evolves along the solution process following a logarithmic curve regurlarly 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. The above predicate is true for each first ant of an iteration, every `iterOnExploit` iterations.

After all ants have built a solution, the local search procedure is applied to all of them.

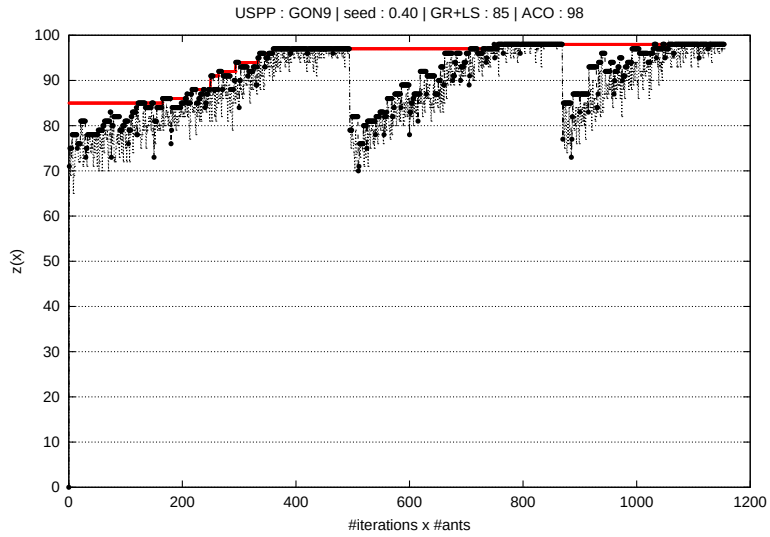


Figure 1: Behavior of the ACO heuristic on the RIC instance 'Gon09'

Among all these solutions, the best one for the current iteration is retained and *evaporation* and *deposition* of pheromones is performed. It means that we increase the pheromones $\phi_i$ corresponding to the items selected in the best packing of the current iteration, whilst we decrease the other pheromones. Besides a disturbance strategy (see figure 1) has been integrated to this management procedure. This strategy is triggered when two conditions are true: (1) the convergence of the ACO is in stagnation, and (2) at least one pheromone has its level set to zero. Finally, the procedure is stopped when the predicate `isFinished?` is true, which occurs when the pheromones are stabilized, after at least two applications of the disturbance.

## 3 Numerical Experiments and Analysis

Implementation of algorithms have been performed with C language. The results were obtained on a Pentium IV 1.8Ghz with 512Mb for all ACO versions. In a previous work, we calculated the optimal solutions with a Cplex solver. As Cplex was not capable of solving all mid and large instances, we consider in this case the best known integer solution which is compared to the heuristics. Roughly speaking, this best solution is taken, for a given instance, as the one returned by GRASP, ACO (former version), Cplex (when time limited) which yields the highest value of the total cost.

Characteristics of the instances and results are given for 12 selected instances in Table 1. For each instance, *#variables* and *#constraints* give respectively the number of variables and the number of constraints. The *Density* column corresponds to the percentage of non-null elements in the constraint matrix. The column *weight* indicates the interval in which the costs $c_i$ are comprised. Notice that instances for which the interval is $[1-1]$ are USPP instances. For 4 ACO versions, we report the average objective function value (column *avg value*) found over 16 runs. We also give the maximum objective function value found (column *best value*) and the average required CPU time (column *CPUt*). A column reports also the average

iteration corresponding to the detection of the best value. Only for the version 2.0, the average number of iteration performed before stopping the procedure (dynamic stopping criterion) is given. All the 64 tested random instances are freely available at `www.univ-valenciennes.fr/ROAD/Delorme/Instances-fr.html`. The 15 RIC instances are private.

No room is available in this abstract for a discussion of results. A deep analysis, all details about algorithmic aspects, and all results over 16 runs of the 79 instances will be provided during the talk. Nevertheless, two important facts deserve to be underlined. First, the new data structure allows now the ACO algorithm to deal with large-size instances (not possible with the former version). But in looking the CPU times between the version 1.1 and 1.2, improvements can be again expected on that matter. Second, the quality of solutions for USPP instances has been clearly improved with the new versions of the algorithm. The new stopping criterion and the new local search procedure is profitable for USPP in general, which is not so obvious for WSPP. Here again, improvements can be expected shortly.

## 4 Acknowledgment

We would like to thank Vincent T'kindt (University of Tours, France) who is our collaborator working on related open questions with this work.

## 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. Gandibleux, X. Delorme and V. T'Kindt. An Ant Colony Algorithm for the Set Packing Problem In M. Dorigo, M. Birattari, Ch. Blum, L. Gambardella, Fr. Mondada, and Th. Stutzle, editors, Ant Colony Optimization and Swarm Intelligence, volume 3172 of *Lecture Notes in Computer Sciences*, pages 49–60. Springer, 2004.

[4] 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.

[5] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Willey-Interscience, New York, 1999.

[6] I.H. Osman and G. Laporte. Metaheuristics : a bibliography. *Annals of Operations Research*, 63:513–623, 1996.

[7] M.W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[8] 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.

Table 1: A selection of instances and results on 16 runs

* : it indicates that we don't know if the best known solution is optimal
□ : new best known value found

| Inst. | Characteristics | | | | | ACO 2.0 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | #variables | #constraints | density | weight | best known value | best value | avg value | CPUt (s) | avg iter bef. 1st best val. | avg #iter performed |
| 100r1 | 100 | 500 | 2.0% | [1-20] | 372 | 372 | 372,00 | 0,88 | 11,06 | 90,50 |
| 100r8 | 100 | 100 | 3.1% | [1-1] | 39 | 39 | 38,94 | 0,56 | 2,06 | 67,50 |
| 200r3 | 200 | 1 000 | 1.0% | [1-20] | 731 | 731 | 721,44 | 5,56 | 45,69 | 118,19 |
| 200r16 | 200 | 600 | 1.0% | [1-1] | 79 | 79 | 78,69 | 4,13 | 6,00 | 67,06 |
| 500r3 | 500 | 2 500 | 0.7% | [1-20] | 776* | 776 | 761,81 | 68,31 | 80,94 | 177,81 |
| 500r14 | 500 | 1 500 | 1.2% | [1-1] | 37* | □ 38 | 37,00 | 42,13 | 12,50 | 67,13 |
| 1000r3 | 1 000 | 5 000 | 0.60% | [1-20] | 661* | 643 | 627,63 | 195,75 | 105,19 | 201,25 |
| 1000r8 | 1 000 | 1 000 | 0.60% | [1-1] | 175* | 175 | 173,06 | 832,75 | 28,25 | 70,38 |
| 2000r7 | 2 000 | 2 000 | 0,56% | [1-20] | 1784* | □ 1796 | 1766,44 | 3048,13 | 462,06 | 645,44 |
| 2000r6 | 2 000 | 2 000 | 2,56% | [1-1] | 9* | 9 | 8,81 | 42,50 | 16,19 | 67,88 |
| gon4 | 1 240 | 23 736 | 0,16% | [1-1] | 94 | 94 | 94,00 | 677,00 | 0(GR+LS) | 69,13 |
| gon9 | 3 720 | 482 887 | 0,05% | [1-1] | 93* | □ 98 | 95,44 | 5039,67 | 60,78 | 80,33 |

| Inst. | ACO 1.2 | | | | ACO 1.1 | | | | ACO 1.0 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | best value | avg value | CPUt (s) | avg iter bef. 1st best val. | best value | avg value | CPUt (s) | avg iter bef. 1st best val. | best value | avg value | CPUt (s) | avg iter bef. 1st best val. |
| 100r1 | 372 | 372,00 | 1,83 | 8,44 | 372 | 372,00 | 2,56 | 10,88 | 372 | 372,00 | 2,62 | 15,00 |
| 100r8 | 39 | 39,00 | 1,50 | 2,25 | 39 | 39,00 | 0,56 | 4,50 | 39 | 38,62 | 0,19 | 24,69 |
| 200r3 | 731 | 725,75 | 7,37 | 53,81 | 731 | 725,75 | 14,56 | 59,69 | 729 | 725,06 | 18,18 | 53,38 |
| 200r16 | 79 | 78,94 | 8,64 | 8,75 | 79 | 78,75 | 13,69 | 20,69 | 79 | 78,31 | 5,18 | 56,88 |
| 500r3 | 776 | 765,88 | 90,98 | 92,06 | 776 | 764,44 | 36,06 | 112,62 | 776 | 771,88 | 104,90 | 71,62 |
| 500r14 | 38 | 37,56 | 136,02 | 65,31 | 38 | 37,56 | 24,69 | 48,88 | 37 | 36,44 | 7,82 | 95,69 |
| 1000r3 | 649 | 632.94 | 199,06 | 119,25 | 661 | 632,50 | 56,56 | 118,12 | 649 | 637,81 | 323,00 | 92,19 |
| 1000r8 | 174 | 173.71 | 2495.71 | 78.71 | 174 | 173,75 | 224,50 | 65,88 | 172 | 171,12 | 29,03 | 166,38 |

ACO 2.0: version 1.2 with dynamic stopping criterion
ACO 1.2: version 1.1 with dynamic memory management
ACO 1.1: version 1.0 with new local search procedures
ACO 1.0: previous published version; total iterations = 200