Combining two Pheromone Structures for Solving the Car Sequencing Problem with Ant Colony Optimization

Christine Solnon

LIRIS CNRS UMR 5205, University of Lyon I
Nautibus, 43 Bd du 11 novembre, 69622 Villeurbanne cedex, France

Abstract

The car sequencing problem involves scheduling cars along an assembly line while satisfying capacity constraints. In this paper, we describe an Ant Colony Optimization (ACO) algorithm for solving this problem, and we introduce two different pheromone structures for this algorithm: the first pheromone structure aims at learning for "good" sequences of cars, whereas the second pheromone structure aims at learning for "critical" cars. We experimentally compare these two pheromone structures, that have complementary performances, and show that their combination allows ants to solve very quickly most instances.

Key words: Ant Colony Optimization, Car Sequencing Problem, Multiple Pheromone Structures
PACS:

1 Introduction

The car sequencing problem involves scheduling cars along an assembly line in order to install options (e.g., sun-roof or air-conditioning) on them. Each option is installed by a different station, designed to handle at most a certain percentage of the cars passing along the assembly line, and the cars requiring this option must be spaced so that the capacity of the station is never exceeded.

This problem is NP-hard [Kis04]. It has been formulated as a constraint satisfaction problem (CSP), and is a classical benchmark for constraint solvers [DSvH88,GW99,Tsa93]. Most of these CSP solvers use a complete tree-search

Email address: christine.solnon@liris.cnrs.fr (Christine Solnon).

approach to explore the search space in a systematic way, until either a solution is found, or the problem is proven to have no solution. In order to reduce the search space, this approach is combined with filtering techniques that propagate capacity constraints to reduce variables' domains. In particular, a dedicated filtering algorithm has been proposed for handling capacity constraints [RP97]. This filtering algorithm is very effective to solve some hardly constrained feasible instances, or to prove infeasibility of some over-constrained instances. However, on some other instances, it cannot reduce domains enough to make complete search tractable.

More recently, [GGP04] has proposed an integer programming model where the objective is to find a sequence that minimizes the number of violated constraints subject to a set of linear integer constraints. This approach makes it possible to rather quickly find solutions to difficult instances that are feasible. However, it requires much longer computation times for reaching best-known solutions for infeasible instances.

Hence, different incomplete approaches have been proposed, that leave out exhaustivity, trying to quickly find approximately optimal solutions in an opportunistic way, e.g., local search [DT99,LLW98,PG02,MH02,GPS03,EGN05], large neighbourhood search [PS04], IDWalk [NTG04], evolutionary algorithms [WT95], or Ant Colony Optimization [Sol00,GGP04].

1.1 Ant Colony Optimization

The basic idea of Ant Colony Optimization (ACO) [DD99,DCG99,DS04] is to model the problem to solve as the search for a minimum cost path in a graph, and to use artificial ants to search for good paths. The behavior of artificial ants is inspired from real ants: artificial ants lay pheromone on components (edges and/or vertices) of the graph and each ant chooses its path with respect to probabilities that depend on pheromone trails that have been previously laid by the colony; these pheromone trails progressively decrease by evaporation. Intuitively, this indirect stigmergetic communication means aims at giving information about the quality of path components in order to attract ants, in the following iterations, towards the corresponding areas of the search space. Indeed, for many combinatorial problems, a study of the search space landscape shows a correlation between solution quality and the distance to optimal solutions [JF95,MF99,SH00].

Artificial ants also have some extra-features that do not find their counterpart in real ants. In particular, they are usually associated with data structures that contain the memory of their previous actions, and they may apply some "daemon" procedures, such as local search, to improve the quality of computed

paths. In many cases, pheromone is updated only after having constructed a complete path, and not during the walk, and the amount of pheromone deposited is usually a function of the quality of the complete path. Finally, the probability for an artificial ant to choose a component often depends not only on pheromones, but also on some problem-specific local heuristics.

The first ant algorithm to be applied to a discrete optimization problem has been proposed by Dorigo in [Dor92]. The problem chosen for the first experiments was the Traveling Salesman Problem and, since then, this problem has been widely used to investigate the solving capabilities of ants [DMC96,DG97]. The ACO metaheuristic, described in [DD99,DCG99,DS04], is a generalization of these first ant based algorithms, and has been successfully applied to different hard combinatorial optimization problems such as quadratic assignment problems [GTD99,MC99], vehicle routing problems [BHS99,GTA99], constraint satisfaction problems [Sol02a], maximum clique problems [SF06], or graph matching problems [SSG05].

1.2 Motivations and overview of the paper

We have proposed in [Sol00] a first ACO algorithm dedicated to permutation constraint satisfaction problems —the solution of which is a permutation of a given tuple of values. Performances of this algorithm have been illustrated, among other problems, on the car sequencing problem. In this first ACO algorithm for the car sequencing problem, pheromone is laid on couples of consecutive cars in order to learn for promising sequences of cars.

Then, in [GPS03], we have proposed and compared different heuristics for solving the car sequencing problem in a greedy randomized way. These heuristics aim at favoring the choice of critical cars during greedy constructions of sequences: these critical cars require hardly constrained options and should be sequenced as soon as possible. We have shown that this very simple greedy approach is able to solve many instances very quickly. We have also shown that these greedy heuristics can be integrated within the ACO algorithm of [Sol00] in a very straightforward way.

These greedy heuristics are very efficient and greatly help ants to solve the car sequencing problem. However, designing such problem-dependent heuristics is a difficult task that requires a good knowledge of the problem to solve. Hence, a main objective of this paper is to answer the following question:

> *Would it be possible to use ACO to identify these critical cars, and solve the car sequencing problem without integrating problem-dependent heuristics?*

To answer this question, we introduce a new pheromone structure, that aims

at identifying critical cars, and we compare it with the greedy heuristic of [GPS03]. We also study the integration of this new pheromone structure with the pheromone structure introduced in [Sol00] and that aims at identifying promising sequences of cars.

The paper is organized as follows. Section 2 formally defines the car sequencing problem we are considering here. Section 3 recalls the basic features of the greedy randomized approach of [GPS03]. Section 4 describes the first pheromone structure introduced in [Sol00] and that aims at identifying promising sequences. Section 5 introduces a new pheromone structure that aims at identifying critical cars and Section 6 shows how these two pheromone structures may be combined. We experimentally compare the different pheromone structures and the greedy heuristic in Section 7, and we compare our results with two recent state-of-the-art local search approaches in Section 8.

2 The Car Sequencing Problem

We consider here the classical car sequencing problem introduced in [DSvH88]. A complete description of this problem and some benchmark instances may be found in CSPlib [GW99]. This problem is a special case of the car sequencing problem proposed by Renault for the ROADEF challenge in 2005 [NC05,SCNA07]. In particular, the challenge problem introduces two different priority levels for capacity constraints, and it introduces paint batching constraints. Considering the classical car sequencing problem allows us to focus on heuristics for dealing with capacity constraints.

2.1 Definition of a Car Sequencing Problem

A car sequencing problem is defined by a tuple (C, O, p, q, r) such that

- $C = \{c_1, .., c_n\}$ is the set of cars to be produced;
- $O = \{o_1, .., o_m\}$ is the set of different options;
- $p : O \rightarrow \mathbb{N}$ and $q : O \rightarrow \mathbb{N}$ define capacity constraints, i.e., for every option $o_i \in O$, each subsequence of q_i consecutive cars on the line must not contain more than p_i cars that require option o_i (see 2.2 for complete definition);
- $r : C \times O \rightarrow \{0, 1\}$ defines option requirements, i.e., for each car $c_i \in C$ and for each option $o_j \in O$, $r_{ij} = 1$ if o_j must be installed on c_i, and $r_{ij} = 0$ otherwise.

2.2 Solution of a Car Sequencing Problem

Solving a car sequencing problem involves finding an arrangement of the cars in a sequence, defining the order in which they will pass along the assembly line, such that the capacity constraints are met. We shall use the following notations to denote and manipulate sequences:

- a *sequence*, noted $\pi =< c_{i_1}, c_{i_2}, \ldots, c_{i_k} >$, is a succession of cars;
- the set of all sequences that may be built with a set of cars C is noted Π_C;
- the *length* of a sequence π, noted $|\pi|$, is the number of cars that it contains;
- the *concatenation* of two sequences π_1 and π_2, noted $\pi_1 \cdot \pi_2$, is the sequence composed of the cars of π_1 followed by the cars of π_2;
- a sequence π_1 is a *subsequence* of another sequence π_2, noted $\pi_1 \subseteq \pi_2$, if there exists two (possibly empty) sequences π_3 and π_4 such that $\pi_2 = \pi_3 \cdot \pi_1 \cdot \pi_4$;
- the *cost* of a sequence π is the number of violated capacity constraints, i.e.,

$$cost(\pi) = \sum_{o_i \in O} \sum_{\substack{\pi_k \subseteq \pi \text{ so that} \\ |\pi_k| = q_i}} violation(\pi_k, o_i)$$

$$\text{where } violation(\pi_k, o_i) = \begin{cases} 0 & \text{if } \sum_{<c_l> \subseteq \pi_k} r_{li} \leq p_i; \\ 1 & \text{otherwise.} \end{cases}$$

We can now define the solution process of a car sequencing problem (C, O, p, q, r) as the search of a minimal cost sequence composed of all the cars to be produced.

3 Greedy randomized construction of sequences

Figure 1 describes a greedy randomized algorithm for constructing sequences: starting from an empty sequence [1], cars are iteratively added at the end of the sequence until all cars have been sequenced. At each step, the set of candidate cars (*cand*) is restricted to the set of cars that introduce the smallest number of new constraint violations (line 4) [2]. To break symetries, we also

[1] The sequence π could be initialized to a non empty sequence in order to take into account the last cars sequenced on the line the previous day, as it is the case in the problem proposed by Renault for the ROADEF challenge.

[2] Note that this elitist strategy, that discards cars introducing more constraint violations, may not be optimal for solving over-constrained instances. To solve such over-constrained instances, it may be preferable not to discard cars introducing more constraint violations but to decrease the probability of selecting them, as proposed in [Sol00,GGP04].

Input: an instance (C, O, p, q, r) of the car sequencing problem
a transition probability function $p : C \times \mathcal{P}(C) \times \Pi_C \rightarrow]0; 1]$
Output: a sequence π that contains each car of C once
1- $\pi \leftarrow <>$
2- **while** $\mid \pi \mid \leq \mid C \mid$ **do**
3- let $C–\pi$ denote the set of cars of C that are not yet sequenced in π
4- $cand \leftarrow \{c_k \in C–\pi | \forall c_j \in C–\pi, cost(\pi. < c_k >) \leq cost(\pi. < c_j >)$ and
5- $(\forall o_i \in O, r_{ki} = r_{ji}) \Rightarrow (k \leq j)$ $\}$
6- choose $c_i \in cand$ w.r.t. probability $p(c_i, cand, \pi)$
7- $\pi \leftarrow \pi \cdot < c_i >$
8- **end while**
9- **return** π

Fig. 1. Greedy randomized construction of a sequence of cars.

restrict the set of candidate cars to cars that require different options (line 5). Then, given this set of candidate cars, the next car is chosen with respect to a transition probability function p: given a candidate car $c_i \in C$, a set of candidate cars $cand \in \mathcal{P}(C)$ (where $\mathcal{P}(C)$ is the set of all subsets of C), and a partial sequence $\pi \in \Pi_C$, this transition function returns the probability of actually choosing c_i. This probability may be defined in different ways, and Sections 4, 5 and 6 propose three different definitions for it, based on different pheromone structures.

In this section, we define the probability transition function proportionally to a heuristic function η that locally evaluates the "hardness" of a candidate car c_i, i.e.,

$$p(c_i, cand, \pi) = \frac{[\eta(c_i, \pi)]^\beta}{\sum_{c_k \in cand} [\eta(c_k, \pi)]^\beta}$$

where β is a numerical parameter which allows one to tune the weight of the heuristic in the transition policy: the higher β, the greedier the policy.

We have introduced and compared in [GPS03] five different definitions for the heuristic function η. These definitions are based on utilization rates of required options and aim at favoring the choice of cars requiring options that have high demands with respect to capacities. The heuristic function of [GPS03] that obtained the best average results is defined by the sum of the utilization rates of the options required by the car, i.e.,

$$\eta(c_i, \pi) = \sum_{o_j \in O} r_{ij} \cdot \text{utilRate}(o_j, C - \pi)$$

where $\text{utilRate}(o_j, C - \pi)$ is the utilization rate of option o_j with respect to the set $C - \pi$ of cars that are not yet sequenced in π. This utilization rate is the

percentage of cars of $C-\pi$ requiring o_j with respect to the maximum number of cars in a sequence of length $|C-\pi|$ which could have o_j while satisfying its capacity constraint, i.e.,

$$utilRate(o_j, C-\pi) = \frac{q_j \cdot \sum_{c_k \in C-\pi} r_{kj}}{p_j \cdot \mid C-\pi \mid}$$

An utilization rate close to 1 (resp. 0) indicates that the demand is very high (resp. low) with respect to the capacity of the station.

4 A first pheromone structure for identifying good car sequences

Solving an instance (C, O, p, q, r) of the car sequencing problem involves finding a permutation of the set of cars C that satisfies capacity constraints. This problem can easily be modelled as the search for a best hamiltonian path in a graph that associates a vertex with each car. Such hamiltonian path finding problems are classical applications for the ACO metaheuristic: for these problems, ants lay pheromone on the graph edges in order to learn for promising sequences of vertices. Based on this principle, we have proposed in [Sol00] a first ACO algorithm for the car sequencing problem, and we briefly recall its main features here.

Basically, the algorithm follows the *MAX-MIN* Ant System scheme [SH00]. First, pheromone trails are initialized to a given upper bound τ_{max_1}. Then, at each cycle every ant constructs a sequence, and pheromone trails are updated. To prevent premature convergence, pheromone trails are bounded within two given bounds τ_{min_1} and τ_{max_1} such that $0 < \tau_{min_1} < \tau_{max_1}$. The algorithm stops iterating either when an ant has found a solution, or when a maximum number of cycles has been performed.

4.1 Pheromone structure

Pheromone is laid on couples of cars. For every couple of different cars $(c_i, c_j) \in C \times C$, we associate a pheromone trail $\tau_1(c_i, c_j)$. Intuitively, this pheromone trail represents the learnt desirability of scheduling c_j just after c_i.

4.2 Construction of sequences by ants

At each cycle every ant constructs a sequence, following the greedy randomized algorithm of Figure 1. To choose the next car c_i to be added at the end of the current sequence π, the transition probability function depends on two factors: a pheromone factor which evaluates the learnt desirability of adding c_i at the end of π, and the heuristic factor η introduced in section 3, i.e.,

$$p(c_i, cand, \pi) = \frac{[\tau_1(c_j, c_i)]^{\alpha_1} \cdot [\eta(c_i, \pi)]^{\beta}}{\sum_{c_k \in cand}[\tau_1(c_j, c_k)]^{\alpha_1} \cdot [\eta(c_k, \pi)]^{\beta}} \text{ if the last car of } \pi \text{ is } c_j$$

$$p(c_i, cand, \pi) = \frac{[\eta(c_i, \pi)]^{\beta}}{\sum_{c_k \in cand}[\eta(c_k, \pi)]^{\beta}} \qquad \text{if } \pi = <>$$

As usually in ACO algorithms, α_1 and β are numerical parameters that are used to determine the relative weights of the two factors.

4.3 Pheromone updating step

Once every ant has constructed a sequence, pheromone trails are updated. First, all pheromone trails are decreased in order to simulate evaporation, i.e., for each couple of different cars $(c_i, c_j) \in C \times C$, the quantity of pheromone $\tau_1(c_i, c_j)$ is multiplied by a factor $(1 - \rho_1)$ where ρ_1 is the evaporation rate such that $0 \leq \rho_1 \leq 1$.

Then, the best ants of the cycle deposit along their paths a trail of pheromone which is inversely proportional to the number of violated constraints: for each sequence π constructed during the cycle, if the cost of π is minimal for this cycle then, for each couple of consecutive cars $< c_j, c_k > \subseteq \pi$, we increment $\tau_1(c_j, c_k)$ by $1/cost(\pi)$.

Finally, pheromone trails that are lower (resp. greater) than τ_{min_1} (resp. τ_{max_1}) are set to τ_{min_1} (resp. τ_{max_1}). The goal of this pheromone bounding step is to favor a better exploration of the search space by preventing the relative differences between pheromone trails from becoming too extreme during processing [SH00].

5 A second pheromone structure for identifying critical cars

The heuristic function η combined with the first pheromone structure in the transition probability function defined in Section 4 aims at favoring the choice

of critical cars, i.e., cars that require options with high utilization rates so that they can hardly be scheduled without violating capacity constraints. We now introduce a new pheromone structure for identifying these critical cars with respect to past experiences of the colony.

5.1 Pheromone structure

Different cars may require a same subset of options, and all these cars are equivalent with respect to the hardness of sequencing them. Hence, we group the cars requiring the same options into classes, and we associate a pheromone structure with every different class.

More formally, we define the class of a car $c_i \in C$ as the set of options required by c_i, i.e., $classOf(c_i) = \{o_j \in O | r_{ij} = 1\}$, and we denote by $classes(C)$ the set of all different car classes, i.e., $classes(C) = \{classOf(c_i) | c_i \in C\}$. Given a car class $cc \in classes(C)$, we note $\tau_2(cc)$ the quantity of pheromone laying on it. Intuitively, this quantity represents the past experience of the colony concerning the difficulty of sequencing cars of this class without violating capacity constraints.

The second pheromone structure introduced here is not managed according to the *MAX-MIN* Ant System of [SH00]. Indeed, imposing lower and upper bounds on pheromone trails and initializing them to the upper bound favor a larger exploration of the search space at the beginning of the search but, as a counterpart, increase the time spent before converging towards good solutions. However, first experiments showed us that it is necessary to quickly identify critical cars to build good solutions: without such identification (e.g., when running the greedy algorithm of Figure 1 with the β parameter set to zero), the constructed sequences violate hundreds of capacity constraints. Hence, to favor a quicker feedback on the critical cars we only introduce a lower bound τ_{min_2} (that ensures that the probability of choosing a car cannot become null), and pheromone trails are initialized to the lower bound τ_{min_2} at the beginning of the search.

5.2 Construction of a sequence by an ant

Ants incrementally build sequences following the algorithm described in Fig. 1. To choose the next car c_i to be added at the end of the current sequence π, the transition probability function only depends on a pheromone factor which

evaluates the learnt hardness of the class of c_i, i.e.,

$$p(c_i, cand, \pi) = \frac{[\tau_2(classOf(c_i))]^{\alpha_2}}{\sum_{c_k \in cand}[\tau_2(classOf(c_k))]^{\alpha_2}}$$

where α_2 is a numerical parameter which allows one to tune the weight of the pheromone factor.

5.3 Pheromone updating step

Ants lay pheromone on car classes during the construction of sequences: each time no more cars can be scheduled without introducing some new constraint violations, some pheromone is added on the classes of the cars that still have to be scheduled (thus indicating that the cars of this class should have been scheduled earlier). The quantity of pheromone added is equal to the number of new constraint violations introduced by the cars of this class. More precisely, we modify the algorithm of Figure 1 by inserting between lines 5 and 6 the following lines:

if $\forall c_i \in cand,\ cost(\pi. < c_i >) > cost(\pi)$ **then**
  for every car class $cc \in \{classOf(c_i) \mid c_i \in C - \pi\}$ **do**
    $\tau_2(cc) \leftarrow \tau_2(cc) + cost(\pi. < c_i >) - cost(\pi)$
    (where c_i is a car of the class cc)

Note that this pheromone laying procedure occurs during the construction step, and not after all ants have completed their construction step, like in most ACO algorithms. Indeed, the quantity of pheromone laid does not depend on the global quality of the sequence but on the local evaluation of the car class with respect to the partial sequence that is currently built. Note also that every ant adds pheromone, and not only the best one(s) of the cycle.

Finally, pheromone is evaporated after every sequence construction. This is done by multiplying the quantity of pheromone $\tau_2(cc)$ laying on each car class $cc \in classes(C)$ by a factor $(1-\rho_2)$ where ρ_2 is the evaporation rate such that $0 \leq \rho_2 \leq 1$.

6 Combining the two pheromone structures

The two proposed pheromone structures achieve two complementary goals: the first one aims at identifying promising subsequences of cars; the second

one aims at identifying critical car classes. Hence, one can easily combine these two complementary pheromone structures:

- Ants lay pheromone on couples of cars $(c_i, c_j) \in C \times C$ and the quantity of pheromone $\tau_1(c_i, c_j)$ represents the past experience of the colony with respect to sequencing car c_j just after car c_i.

  For this first pheromone structure, pheromone trails are bounded within the interval $[\tau_{min_1}; \tau_{max_1}]$ and they are initialized to the upper bound τ_{max_1}. Pheromone trails are updated at the end of each cycle, once every ant of the colony has computed a complete sequence, and only the best ants of the cycle lay pheromone.
- Ants also lay pheromone on car classes $cc \in \mathit{Classes}(C)$ and the quantity of pheromone $\tau_2(cc)$ represents the past experience of the colony with respect to the difficulty of sequencing cars of this class without violating constraints.

  For this second pheromone structure, a lower bound τ_{min_2} is imposed and pheromone trails are initialized to this lower bound. Pheromone is laid by every ant while it constructs a sequence, and the evaporation step occurs at the end of every sequence construction by an ant.

The algorithm followed by ants to build sequences is the same as the one displayed in figure 1. To choose the next car c_i to be added at the end of the current sequence π, the transition probability function depends on two different pheromone factors, i.e.,

$$p(c_i, cand, \pi) = \frac{[\tau_1(c_j, c_i)]^{\alpha_1} \cdot [\tau_2(classOf(c_i))]^{\alpha_2}}{\sum_{c_k \in cand} [\tau_1(c_j, c_k)]^{\alpha_1} \cdot [\tau_2(classOf(c_k))]^{\alpha_2}} \text{ if the last car of } \pi \text{ is } c_j$$

$$p(c_i, cand, \pi) = \frac{[\tau_2(classOf(c_i))]^{\alpha_2}}{\sum_{c_k \in cand} [\tau_2(classOf(c_k))]^{\alpha_2}} \qquad \text{if } \pi = <>$$

As usual, α_1 and α_2 are numerical parameters that are used to determine the relative weights of the two pheromone factors.

Note that the greedy heuristic function η introduced in section 3, and combined with the first pheromone structure in section 4 is no longer used in this new transition probability. Indeed, this heuristic factor, that aims at identifying critical cars, has been replaced by the second pheromone structure.

7 Experimental comparison of the pheromone structures and the greedy heuristic

7.1 Considered algorithms and test suites

To compare the pheromone structures τ_1 and τ_2 and the greedy heuristic function η, we consider four different versions of the greedy randomized algorithm of Figure 1, based on four different definitions of the transition probability function p:

- in *Greedy(η)* the transition probability function is based on the greedy heuristic function η as described in Section 3;
- in *ACO(τ_1, η)* the transition probability function is based on a combination of the first pheromone structure τ_1 and the greedy heuristic function η as described in Section 4;
- in *ACO(τ_2)* the transition probability function is based on the second pheromone structure τ_2 as described in Section 5;
- in *ACO(τ_1, τ_2)* the transition probability function is based on a combination of the two pheromone structures τ_1 and τ_2 as described in Section 6.

All algorithms have been implemented in C and run on a 2GHz Pentium 4.

These four different versions are experimentally compared on a test suite of 82 instances generated by Perron and Shaw and described in [PS04]. In this test suite, all instances have $|O| = 8$ options and $|Classes(C)| = 20$ different car classes; capacity constraints are randomly generated while ensuring that $\forall o_i \in O, 1 \leq p_i \leq 3$ and $p_i < q_i \leq p_i + 2$. The number of cars $|C|$ to be sequenced varies between 100 and 500: 32 (resp. 21 and 29) instances have 100 (resp. 300 and 500) cars. All instances are feasible and have at least one solution that satisfies all capacity constraints.

7.2 Parameter settings

Each run of each considered algorithm is limited to 150000 constructions of sequences: for *ACO(τ_1, η)* and *ACO(τ_1, τ_2)* we have fixed the maximum number of cycles to 5000 and the number of ants to 30; for *Greedy(η)* and *ACO(τ_2)* we have fixed the maximum number of cycles to 150000 as a single sequence is built at each cycle. Note that the construction of 150000 sequences roughly corresponds to 20 (resp. 50 and 100) seconds of CPU time on a 2GHz Pentium 4 for instances with 100 (resp. 300 and 500) cars.

The setting of the other parameters is summarized in Table 1 and is discussed

Table 1
Parameter settings.

	Heuristic η	1^{st} pheromone structure				2^{nd} pheromone structure		
	β	α_1	ρ_1	τ_{min_1}	τ_{max_1}	α_2	ρ_2	τ_{min_2}
Greedy(η)	6	-	-	-	-	-	-	-
ACO(τ_1, η)	6	2	1%	0.01	4	-	-	-
ACO(τ_2)	-	-	-	-	-	6	3%	1
ACO(τ_1, τ_2)	-	2	1%	0.01	4	6	3%	1

below. To tune parameters, we have run algorithms on a subset of the test suite instances that contains 32 instances (9 instances with 100 cars, 7 instances with 300 cars and 16 instances with 500 cars). These 32 instances have been selected for tuning parameters because they appeared to be the most difficult ones for the greedy algorithm (other instances were trivially solved).

Both *Greedy(η)* and *ACO(τ_1, η)* use the greedy heuristic η in their probability transition function and have to set the parameter β that determines the weight of this heuristic. Experiments have been done with different values for β, ranging between 1 and 10. The best average results have been obtained when it is fixed to 6 (eventhough rather equivalent results were obtained when $\beta \in [4; 8]$).

The three ACO variants use the pheromone structures τ_1 and/or τ_2 in their probability transition functions and have to set the pheromone factor weight α, the pheromone evaporation rate ρ, and the bounds τ_{min} and τ_{max} associated with these pheromone structures. Setting these parameters makes it possible to balance between two dual goals when exploring the search space: on the one hand, one has to *intensify* the search around the most "promising" areas, that are usually close to the best solutions found so far; on the other hand, one has to *diversify* the search and favor exploration in order to discover new, and hopefully more successful, areas of the search space. Diversification may be emphasized in ACO algorithms by decreasing α —so that ants become less sensitive to pheromone — or by decreasing ρ —so that pheromone evaporates more slowly— or by decreasing the difference between τ_{min} and τ_{max} —so that the relative difference between pheromone factors decreases. When increasing the exploratory ability of ants in this way, one usually finds better solutions, but as a counterpart it takes longer to find them. This duality has been observed on many different combinatorial problems such as, e.g., maximum clique problem [SF06] or constraint satisfaction problem [Sol02b].

We have chosen two different parameter settings for the two pheromone structures τ_1 and τ_2. Indeed, as pointed out in Section 5, the identification of critical cars appears to be essential for building good solutions, so that one has to set

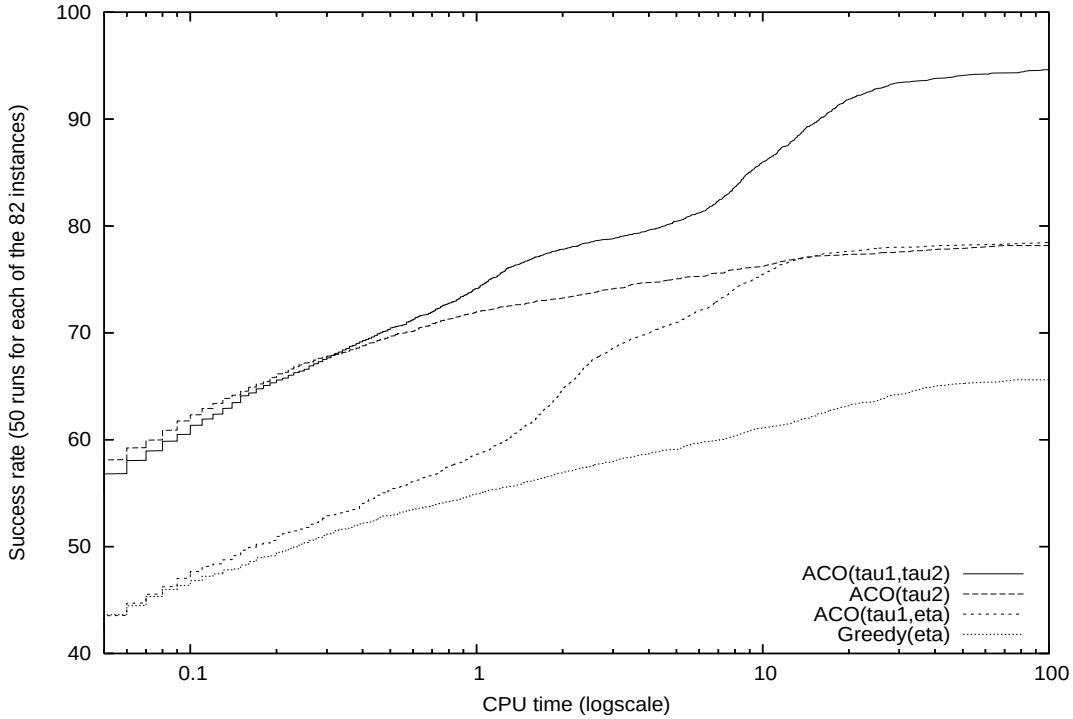

Fig. 2. Comparison of *Greedy(η)*, *ACO(τ_1, η)*, *ACO(τ_2)*, and *ACO(τ_1, τ_2)*. Each curve plots the evolution of the percentage of successful runs (over 50 runs on each of the 82 instances) with respect to CPU time (in logscale).

α_2 and ρ_2 to values that favor intensification in order to have a quick feedback, i.e., $\alpha_2 = 6$ and $\rho_2 = 0.03$. On the contrary, we have chosen a setting that ensures a good diversification of the search for the first pheromone structure, i.e., $\alpha_1 = 2$ and $\rho_1 = 0.01$ (as discussed in [GPS03]).

Note that the performances of the three ACO variants are rather stable with respect to these pheromone parameter settings. For example, when running *ACO(τ_1, τ_2)* with different values for α and ρ (with $\alpha_1 \in \{1, 2, 3, 4\}$, $\rho_1 \in \{0.5\%, 1\%, 2\%, 3\%\}$, $\alpha_2 \in \{4, 6, 8, 10\}$, and $\rho_2 \in \{1\%, 2\%, 3\%, 5\%\}$), the final success rates (over 50 runs on each of the 82 instances) vary between 85.3% for the worth configuration and 94.7% for the best one, whereas the average is 91.3%.

7.3 Experimental comparison

Figure 2 displays the evolution of the percentage of successful runs (that have found a solution) of the four considered algorithms with respect to CPU time.

Let us first compare *Greedy(η)* with *ACO(τ_2)*, as both algorithms build sequences with respect to a greedy heuristic that evaluates the hardness of car classes, but differ on the way this estimation is done: *Greedy(η)* considers the

sum of the utilization rates of the required options, whereas *ACO(τ_2)* "learns" this with respect to past experiences of the colony. Figure 2 shows that the success rate of *ACO(τ_2)* is always more than 10% as high as the success rate of *Greedy(η)*: after 0.05 seconds of CPU-time, *Greedy(η)* and *ACO(τ_2)* have respectively solved 42.5% and 55.8% of the runs, whereas after 100 seconds of CPU-time, they have respectively solved 65.6% and 77.6% of the runs.

Let us then compare *Greedy(η)* with *ACO(τ_1, η)* (resp. *ACO(τ_2)* with *ACO-(τ_1,τ_2)*) to evaluate the benefit of integrating the first pheromone structure τ_1 within the transition probability function. Figure 2 shows that at the beginning of the solution process, the success rates of *Greedy(η)* and *ACO(τ_1, η)* (resp. *ACO(τ_2)* and *ACO(τ_1, τ_2)*) are very close. Indeed, the setting of the parameters for managing τ_1 has been chosen in order to favor exploration so that during the first cycles τ_1 does not significantly influence transition probabilities: at the beginning of the search process, pheromone trails are initialized to the upper bound, τ_{max_1}, so that after k cycles, the quantity of pheromone is bounded between $\tau_{max_1} \cdot (1 - \rho_1)^k$ and τ_{max_1}. Then, after five hundred or so cycles (roughly corresponding to two, five, and ten seconds for instances with 100, 300 and 500 cars respectively), the first pheromone structure actually influences ants so that the success rate of *ACO(τ_1, η)* (resp. *ACO(τ_1, τ_2)*) becomes significantly higher than the success rate of *Greedy(η)* (resp. *ACO(τ_2)*).

As a conclusion, let us note that the second pheromone structure τ_2 allows ants to solve very quickly (in less than 0.05 seconds) more than half of the runs. The first pheromone structure τ_1 needs more time to guide ants towards better sequences (around two, five and ten seconds for instances with 100, 300, and 500 cars respectively), but actually improves the solution process so that, at the end of the solution process, the combination of the two pheromone structures allows ants to solve nearly 95% of the runs.

8 Comparison with other approaches

8.1 Considered approaches

We now compare the best performing of our four variants, *ACO(τ_1, τ_2)*, with two recent local search approaches, i.e., *IDWalk* and *VFLS*.

IDWalk [NTG04] is a new metaheuristic based on local search that appears to obtain better results than other metaheuristics —such as tabu search and simulated annealing— for solving the car sequencing problem among other problems. Besides the maximum number of moves (*MaxMv*), this approach introduces only one parameter, called *Max*, that determines the maximum

number of neighbours that are considered before performing every move. At each iteration *IDWalk* chooses the first non decreasing neighbour, and if all the *Max* considered neighbours deteriorate the current solution, *IDWalk* chooses the best one over them. The *Max* parameter is automatically determined at the beginning of the search by performing a few short walks (of 2000 moves) with different possible values. This value is also reactively adapted during the solution process when the length of the walk reaches some given limits. In the case of the car sequencing problem, the neighbourhood is defined by the set of all sequences that can be obtained by swaping any pair of cars that have different car classes and such that at least one car is involved in one or more constraint violations; the car that is involved in constraint violations is chosen with respect to a probability that is proportional to the number of constraints it violates. The initial sequence of cars from which the local search is performed is constructed in a greedy way, using the η heuristic described in Section 3.

VFLS (for "Very Fast Local Search") [EGN05,EGN07] is the algorithm that won the ROADEF challenge on the car sequencing problem [NC05,SCNA07]. In addition to capacity constraints, the ROADEF problem contains some extra constraints that are related to car colors and that aim at minimizing solvents. The implementation of *VFLS* considered in this comparison is an adaptation of the one that have won the challenge, where data structures have been optimized to focus on capacity constraints only. The *VFLS* algorithm is based on a local search approach similar to the one proposed in [GPS03]: starting from an initial sequence that is constructed in a greedy way using a heuristic based on utilization rates, the algorithm chooses at each iteration the first neighboor of the current solution that does not increase its cost. The considered neighbourhood is defined by a set of five transformations (exchange of two cars, insertion of a car backward or forward, and mirror and random permutations of subsequences). The probability of choosing each of these transformations respectively is 0.6, 0.13, 0.13, 0.13, and 0.01.

8.2 Test suite and experimental setup

We compare the three approaches on the 82 instances of the test suite provided by Perron and Shaw. However, to discuss scale-up properties, we have gouped these instances into three subsets with respect to the number of cars $|C|$ to be sequenced: the first group contains the 32 instances with 100 cars; the second one contains the 21 instances with 300 cars; and the third one contains the 29 instances with 500 cars.

The three algorithms have been implemented in C or C++, and have been

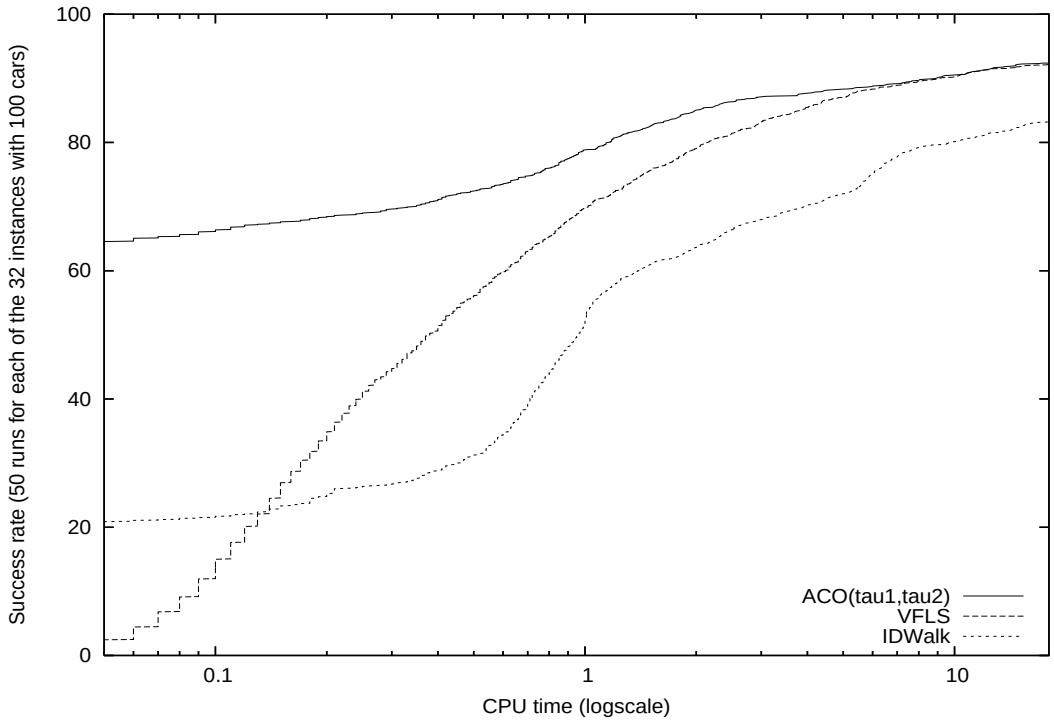

Fig. 3. Evolution of the percentage of successful runs with respect to CPU time for $ACO(\tau_1, \tau_2)$, VFLS, and IDWalk, on the 32 instances with 100 cars.

run on the same 2GHz Pentium 4 [3]. Each algorithm has been run 50 times on each instance.

8.3 Experimental Results

Figures 3, 4, and 5 show the evolution of the success rates with respect to CPU time on instances with 100, 300, and 500 cars respectively. On the three figures, we note that for very short CPU time limits $ACO(\tau_1, \tau_2)$ is better than *IDWalk*, which itself is better than *VFLS*: after 0.1 second of CPU time, $ACO(\tau_1, \tau_2)$ has solved more than 60% of the runs while *VFLS* and *IDWalk* have respectively solved 6% and 25% of the runs. However, the success rate of *VFLS* increases more steeply than the success rates of *IDWalk* and $ACO(\tau_1, \tau_2)$. Therefore, after a few seconds of CPU time, *VFLS* has clearly surpassed *IDWalk*. Finally, at the end of the processing time, *VFLS* has reached the success rate of $ACO(\tau_1, \tau_2)$ for the instances with 100 and 300 cars, whereas *VFLS* has outperformed $ACO(\tau_1, \tau_2)$ for the instances with 500 cars.

The fact that $ACO(\tau_1, \tau_2)$ is outperformed by *VFLS* at the end of the processing time for the largest instances that have 500 cars brings to the fore the

[3] The code of *IDWalk* and *VFLS* have been kindly sent by their authors, and the obtained results have been validated by the authors.

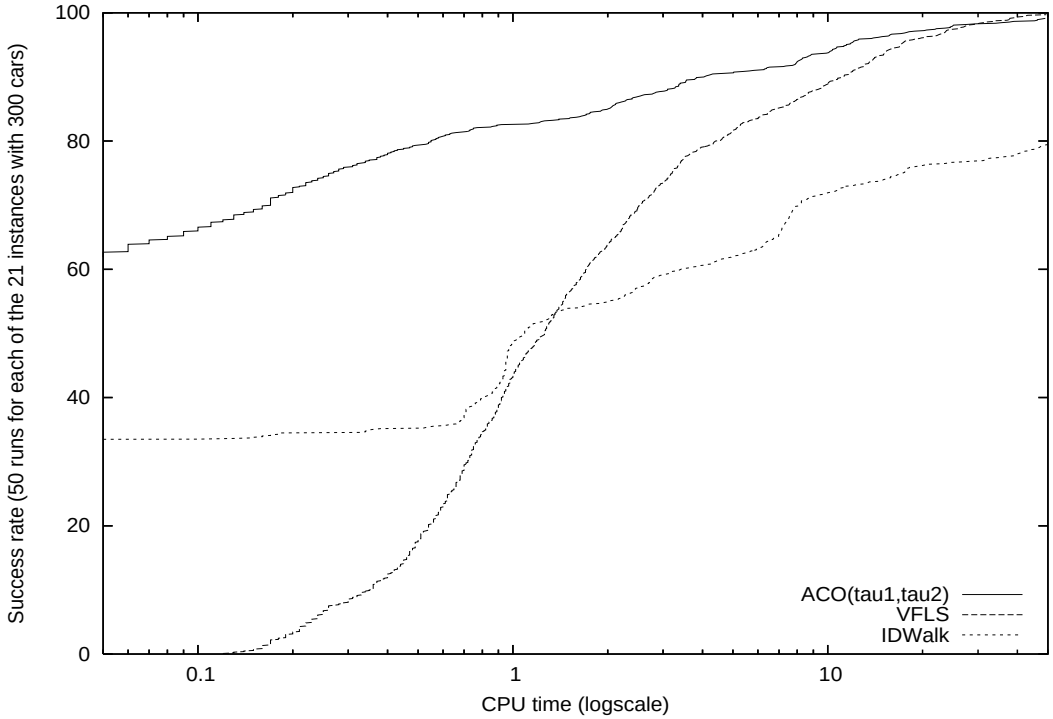

Fig. 4. Evolution of the percentage of successful runs with respect to CPU time for $ACO(\tau_1, \tau_2)$, VFLS, and IDWalk, on the 21 instances with 300 cars.

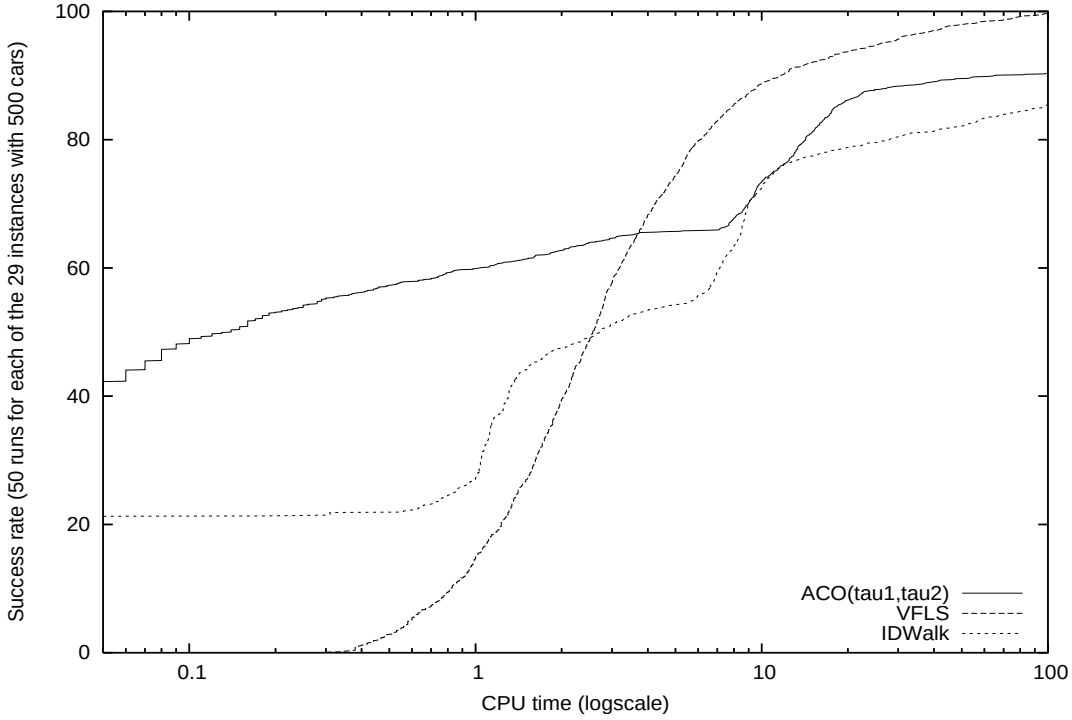

Fig. 5. Evolution of the percentage of successful runs with respect to CPU time for $ACO(\tau_1, \tau_2)$, VFLS, and IDWalk, on the 29 instances with 500 cars.

differences in the time complexities of these algorithms. Indeed, if there are $|C| = n$ cars to sequence and $|classes(C)| = k$ different car classes, then the complexity of constructing one sequence by an ant is in $O(nk)$, the complexity of the pheromone laying step is in $O(n)$ and the complexity of the evaporation step is in $O(n^2)$. As a comparison, the time complexity of performing one move, in both *IDWalk* and *VFLS*, does not depend on the total number of cars to be sequenced: each move is locally evaluated by considering only a subsequence of cars and the time complexity of one move mainly depends on the sizes (q_i) of the gliding subsequences on which the capacity constraints must be checked. In the test suite we have considered here, these sizes are bounded between 2 and 5 for all instances.

9 Conclusion

We have introduced a new pheromone structure for solving the car sequencing problem. This pheromone structure aims at identifying critical cars and experimental results have shown us that it obtains better results than the greedy heuristic based on utilization rates. We have also shown that this new pheromone structure may be combined with another pheromone structure that aims at identifying good subsequences of cars.

This double pheromone structure may be related to previous work on multiple pheromone matrices for solving other combinatorial optimization problems. In particular, [IMM01] introduces two different pheromone structures to solve bi-criteria optimization problems, where each pheromone structure is tailored to one optimization criteria. [MR02] also introduces two different pheromone structures, one for intensifying the search around good solutions, and one for diversifying the search around apparently poorer areas of the search space. Multi colony approaches such as, e.g., [MRS02,CRP04], also consider multiple pheromone matrices. However, in these approches the different pheromone matrices have an homogeneous structure, as these different pheromone matrices are used by different ant colonies that work in parallel to solve a combinatorial optimization problem.

We have shown in this paper that the combination of two complementary pheromone structures makes it possible to obtain very competitive results on the car sequencing problem, being able to solve many instances much quicker than *VFLS*, the local search based algorithm that won the ROADEF 2005 challenge. However, on the largest instances and for longer time limits, ACO is outperformed by *VFLS*. Hence, further work will mainly concern the integration of local search techniques within our "double" ACO algorithm. Indeed, local search may be combined with the ACO metaheuristic in a very straightforward way: ants construct solutions exploiting pheromone, and local search

improves their quality by iteratively performing local moves. Actually, the best-performing ACO algorithms for many combinatorial optimization problems are hybrid algorithms that combine probabilistic solution construction by a colony of ants with local search [DG97,SH00,Sol02a,SF06,SSSG06].

Finally, even if it is not a criteria for comparing approaches, let us point out the simplicity of our double ACO algorithm which is very easy to implement: the C code corresponding to $ACO(\tau_1, \tau_2)$ contains less than 300 lines (including input/output issues). This is generally not the case of local search approaches which have to use more sophisticated data structures in order to incrementally evaluate moves.

Acknowledgements. Many thanks to Bertrand Estellon, Frédéric Gardi and Karim Nouioua for giving the code of *VFLS*, and to Bertrand Neveu and Gilles Trombettoni for giving the code of *IDWalk*.

References

[BHS99] B. Bullnheimer, R.F. Hartl, and C. Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.

[CRP04] S.C. Chu, J.F. Roddick, and J.S. Pan. Ant colony system with communication strategies. *Information Sciences*, 167:63–76, 2004.

[DCG99] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[DD99] M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, UK, 1999.

[DG97] M. Dorigo and L.M. Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[DMC96] M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 26(1):29–41, 1996.

[Dor92] M. Dorigo. *Optimization, Learning and Natural Algorithms* (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[DS04] M. Dorigo and T. Stuetzle. *Ant Colony Optimization*. MIT Press, 2004.

[DSvH88] M. Dincbas, H. Simonis, and P. van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In Y. Kodratoff, editor, *Proceedings of ECAI-88*, pages 290–295, 1988.

[DT99] A.J. Davenport and E.P.K. Tsang. Solving constraint satisfaction sequencing problems by iterative repair. In *Proceedings of the first international conference on the practical applications of constraint technologies and logic programming (PACLP)*, pages 345–357, 1999.

[EGN05] B. Estellon, F. Gardi, and K. Nouioua. Ordonnancement de véhicules : une approche par recherche locale à grand voisinage. In *Actes des premières Journées Francophones de Programmation par Contraintes (JFPC)*, pages 21–28, 2005.

[EGN07] B. Estellon, F. Gardi, and K. Nouioua. Real-life car sequencing: very large neighborhood search vs very fast local search. *European Journal of Operational Research (EJOR)*, 2007.

[GGP04] M. Gravel, C. Gagné, and W.L. Price. Review and comparison of three methods for the solution of the car-sequencing problem. *Journal of the Operational Research Society*, 2004.

[GPS03] J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search and ant colony optimization approaches for car sequencing problems. In *Applications of evolutionary computing*, volume 2611 of *LNCS*, pages 246–257. Springer, 2003.

[GTA99] L.M. Gambardella, E.D. Taillard, and G. Agazzi. MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw Hill, London, UK, 1999.

[GTD99] L.M. Gambardella, E. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50:167–176, 1999.

[GW99] I.P. Gent and T. Walsh. Csplib: a benchmark library for constraints. Technical report, APES-09-1999, 1999. available from http://csplib.cs.strath.ac.uk/. A shorter version appears in CP99.

[IMM01] S. Iredi, D. Merkle, and M. Middendorf. Bi-criterion optimization with multi-colony ant algorithms. In *First International Conference on Evolutionary Multi-Criterion Optimization*, number 1993 in Lecture Notes in Computer Science, pages 359–372. Springer, 2001.

[JF95] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of International Conference on Genetic Algorithms, Morgan Kaufmann, Sydney, Australia*, pages 184–192, 1995.

[Kis04] T. Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32:331–335, 2004.

[LLW98] J.H.M. Lee, H.F. Leung, and H.W. Won. Performance of a comprehensive and efficient constraint library using local search. In *11th Australian JCAI*, LNAI. Springer-Verlag, 1998.

[MC99] V. Maniezzo and A. Colorni. The Ant System applied to the quadratic assignment problem. *IEEE Transactions on Data and Knowledge Engineering*, 11(5):769–778, 1999.

[MF99] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 245–260. McGraw Hill, UK, 1999.

[MH02] L. Michel and P. Van Hentenryck. A constraint-based architecture for local search. In *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 83–100, New York, NY, USA, 2002. ACM Press.

[MR02] J. Montgomery and M. Randall. Ant-pheromones as a tool for better exploration of search space. In *Third International Workshop on Ants Algorithms (ANTS'2002)*, number 2463 in Lecture Notes in Computer Science, pages 100–110. Springer, 2002.

[MRS02] M. Middendorf, F. Reischle, and H. Schmeck. Multi colony ant algorithms. *Journal of Heuristics*, 8(3):305–320, 2002.

[NC05] A. Nguyen and V.-D. Cung. Le problème du car sequencing renault et le challenge roadef'2005. In *Premières Journées Francophones de Programmation par Contraintes (JFPC 2005)*, pages 3–10, 2005.

[NTG04] B. Neveu, G. Trombettoni, and F. Glover. Id walk: A candidate list strategy with a simple diversification device. In *Proceedings of CP'2004*, volume 3258 of *LNCS*, pages 423–437. Springer Verlag, 2004.

[PG02] M. Puchta and J. Gottlieb. Solving car sequencing problems by local optimization. In *Applications of Evolutionary Computing (EvoCOP 2002)*, volume 2279 of *LNCS*, pages 132–142. Springer, 2002.

[PS04] L. Perron and P. Shaw. Combining forces to solve the car sequencing problem. In *Proceedings of CP-AI-OR'2004*, volume 3011 of *LNCS*, pages 225–239. Springer, 2004.

[RP97] J.-C. Regin and J.-F. Puget. A filtering algorithm for global sequencing constraints. In *CP97*, volume 1330 of *LNCS*, pages 32–46. Springer-Verlag, 1997.

[SCNA07] C. Solnon, V.-D. Cung, A. Nguyen, and C. Artigues. Editorial: The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the roadef'2005 challenge problem (to appear). *European Journal of Operational Research (EJOR)*, 2007.

[SF06] C. Solnon and S. Fenet. A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3):155–180, 2006.

[SH00] T. Stützle and H.H. Hoos. MAX-MIN Ant System. *Journal of Future Generation Computer Systems, special issue on Ant Algorithms*, 16:889–914, 2000.

[Sol00] C. Solnon. Solving permutation constraint satisfaction problems with artificial ants. In *Proceedings of ECAI'2000, IOS Press, Amsterdam, The Netherlands*, pages 118–122, 2000.

[Sol02a] C. Solnon. Ants can solve constraint satisfaction problems. *IEEE Transactions on Evolutionary Computation*, 6(4):347–357, 2002.

[Sol02b] C. Solnon. Boosting ACO with a preprocessing step. In *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*, volume 2279 of *LNCS*, pages 161–170. Springer-Verlag, 2002.

[SSG05] O. Sammoud, C. Solnon, and K. Ghédira. Ant algorithm for the graph matching problem. In Springer Verlag, editor, *5th European Conf. on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, volume LNCS 3448, pages 213–223, 2005.

[SSSG06] O. Sammoud, S. Sorlin, C. Solnon, and K. Ghédira. A Comparative Study of Ant Colony Optimization and Reactive Search for Graph Matching Problems. In G. Raidl and J. Gottlieb, editors, *6th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006)*, LNCS, pages 287–301. Springer, 2006.

[Tsa93] E.P.K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, UK, 1993.

[WT95] T. Warwick and E. Tsang. Tackling car sequencing problems using a genetic algorithm. *Journal of Evolutionary Computation - MIT Press*, 3(3):267–298, 1995.