Optimization of a Real-World Auto-Carrier Transportation Problem

Mauro Dell'Amico, Simone Falavigna, Manuel Iori

To cite this article:
Mauro Dell'Amico, Simone Falavigna, Manuel Iori (2014) Optimization of a Real-World Auto-Carrier Transportation Problem. Transportation Science

Published online in Articles in Advance 12 Mar 2014

http://dx.doi.org/10.1287/trsc.2013.0492

Full terms and conditions of use: http://pubsonline.informs.org/page/terms-and-conditions

This article may be used only for the purposes of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval. For more information, contact permissions@informs.org.

The Publisher does not warrant or guarantee the article’s accuracy, completeness, merchantability, fitness for a particular purpose, or non-infringement. Descriptions of, or references to, products or publications, or inclusion of an advertisement in this article, neither constitutes nor implies a guarantee, endorsement, or support of claims made of that product, publication, or service.

Copyright © 2014, INFORMS

Please scroll down for article—it is on subsequent pages

INFORMS is the largest professional society in the world for professionals in the fields of operations research, management science, and analytics.
For more information on INFORMS, its publications, membership, or meetings visit http://www.informs.org
We study a real-world distribution problem arising in the automotive field in which cars, trucks, and other vehicles have to be loaded onto auto-carriers and then delivered to dealers. The solution of the problem involves both the computation of the routing of the auto-carriers along the road network and the determination of a feasible loading for each carrier. We solve the problem by means of an iterated local search algorithm that makes use of several inner local search strategies for the routing part and mathematical modeling and enumeration techniques for the loading part. Extensive computational results on real-world instances show that good savings on the total cost can be obtained within small computational efforts.

Keywords: vehicle routing; loading; auto-carrier; iterated local search

History: Received: August 2011; revisions received: August 2012, March 2013; accepted: July 2013. Published online in Articles in Advance.

1. Introduction
The automotive industry is a crucial part of modern economies, as confirmed by gross sales (710 billion euro in 2008) and number of employees (2.3 million direct and 10 million indirect employees in Europe in 2008). Overall, 334.8 million vehicles were circulating on roads in Europe in 2008, 283.2 million in the United States, Canada, and Mexico, and 912.7 million in the entire world (see ANFIA 2010, EUROSTAT 2010).

Despite the 2008 world economic crisis, the number of vehicles sold remains very high. In 2009 in Europe, there were 18.4 million registrations of new motor vehicles, consisting of two million industrial and commercial vehicles and 16.4 million cars. In the same year, North America counted 12.8 million new motor vehicle registrations, including 6.2 million industrial and commercial vehicles and 6.6 million cars. Very large increases were noticed in Brazil, Russia, India, and China.

The large quantities of vehicles sold and the large market fluctuations make optimization techniques suitable for this sector. A good opportunity for optimization derives from the delivery of vehicles to dealers, which represents one of the most important logistic issues in this sector (the European Commission reports that logistics account for about 10% of the cost of the finished product, see European Commission, Mobility and Transport 2006). This is the topic addressed in this paper.

Vehicle manufacturers do not usually deliver their products directly, but rely on logistics companies. These companies receive the vehicles from the manufacturers, stock them in storage areas and deliver them to the dealers when ordered. The deliveries are provided by special trucks, called auto-carriers or auto transporters, usually composed of a tractor and perhaps a trailer, both equipped with upper and lower loading platforms. A typical European auto-carrier with four loading platforms carrying seven vehicles is depicted in Figure 1. For standard auto-carriers, the number of loading platforms is usually between one and four, with four being the most common. The loading shown in Figure 1 is composed of identical vehicles, but in most cases loadings involve a variety of vehicles.

The loading capacity of an auto-carrier strongly depends on the vehicles’ weight and dimensions. The length, height, and shape of the vehicles are important parameters, whereas the width is negligible because vehicles cannot be transported side by side on the auto-carriers. To increase their capacity, auto-carriers are usually equipped with particular loading equipment. For example, the upper loading platforms may be translated vertically and/or rotated (see, e.g., the upper front platform in Figure 1). Both upper and lower platforms can also be extended to increase their lengths. Auto-carriers are loaded from the rear, and unloading without reshuffling the cargo is usually a must; i.e., last-in-first-out (LIFO) policy is imposed. Simply speaking, in Figure 1 the first vehicle to be unloaded is the one in the rear bottom part of the cargo.
Additional loading constraints that must be taken into account come from transportation laws that impose, e.g., maximum length, height, and weight limits. These laws vary from one nation to another (see, e.g., the U.S. Department of Transportation 2004).

The problem of loading the vehicles into an auto-carrier can thus be seen as a particular two-dimensional loading problem with a large number of complicating constraints; hence, its solution is far from trivial.

For routing, dealers are usually spread out over large areas, and a single dealer order can rarely exactly fill the capacity of an integer number of auto-carriers. For this reason, the companies are forced to load orders from different dealers into the same auto-carrier. As a consequence, routing auto-carriers in an optimal way is also not simple.

In this paper we describe the algorithm we developed for the real-world problem of a logistics company that is one of the leaders in the Italian market of vehicle delivery. The core problem can be described as follows:

Given a heterogeneous fleet of auto-carriers based at a central depot and a set of dealers each requiring a set of vehicles, load the vehicles into the auto-carriers and route the auto-carriers along the road network to serve all dealers with minimum cost. Split deliveries are allowed, LIFO policy is imposed, and the cost is given by the total number of kilometers traveled.

The resulting combinatorial problem is challenging, as it combines two NP-hard problems from the domains of loading and routing. In addition, the size of the problems we address is very large: in the instances provided to us, on average, about 800 vehicles are delivered to 200 dealers using 80 auto-carriers. Since state-of-the-art exact algorithms for the simpler capacitated vehicle-routing problem (CVRP) need days of computations to solve instances with just 100 customers, and since the logistics company requires solutions to be found in minutes, we decided to focus on a heuristic approach.

We developed an iterated local search algorithm that uses eight inner local search techniques. Any time one of these techniques has to determine the feasibility of the loading associated with a route, it invokes a loading algorithm. Such algorithm is based on an approximated modeling of the original two-dimensional problem that is solved by means of an implicit enumeration technique.

The contribution of the paper is thus two-fold: we first provide a mathematical model and an algorithm for the solution of the loading subproblem, and then provide a nontrivial heuristic for the solution of the overall problem. The resulting algorithm is the first approach in the literature that provides detailed solutions of both the loading and routing components of the auto-carrier transportation at the same time. It outputs a precise vehicle-platform assignment to obtain a feasible loading and precise sequence of deliveries for each auto-carrier. It contains features that exploit the complicated structure of the problem; their effectiveness is shown by means of extensive tests on real-world instances. Despite the fact that we focus on the Italian market, the methodology we present is easily replicable in other markets.

The paper is structured as follows. In §2 we formally describe the problem and in §3 we review the relevant literature. The solution of the loading subproblem is described in §4 and that of the overall problem in §5. In §6 we present extensive tests on real-world instances. The effectiveness of our algorithms is shown by comparison with solutions obtained by the algorithm currently used by the logistics company, and solutions obtained under different algorithmic configurations. We also gain insight into the difficulty of the problem by discussing the impact of the different real-world requirements. Finally, in §7 we draw some conclusions.

2. Problem Description
In the following we use the term vehicle to denote a transported item (e.g., a car, a truck, or a van), the term auto-carrier to denote a truck transporting vehicles, and the term dealer to denote a delivery point (i.e., a customer requiring one or more vehicles). We are given a quite involved input, which can be described as follows:

Network: Given is a complete graph $G = (N, E)$, where $N = \{0, 1, \ldots, n\}$ is the set of vertices and $E$ the set of edges connecting each vertex pair. Vertex 0 corresponds to
the depot, whereas vertices \{1, \ldots, n\} correspond to the \(n\) dealers to be served. The edge connecting vertices \(i\) and \(j\) is denoted by \((i, j)\) and has an associated routing cost \(c_{ij}\) \((i, j \in N)\). The cost matrix is symmetric and satisfies the triangular inequality.

**Fleet:** Given is a heterogeneous fleet of auto-carriers, composed by a set \(T\) of auto-carrier types. Each auto-carrier type \(t\) \((t \in T)\) has a maximum weight capacity \(W_t\) and is formed by \(P_t\) loading platforms. There are \(K_t\) auto-carriers available for each type \(t\).

**Demands:** The demand of dealer \(i\) consists of a set \(M_i\) of vehicles \((i \in N \setminus \{0\})\). Each vehicle \(k \in M_i\) demanded by dealer \(i\) belongs to a vehicle type (or vehicle model), which is defined by a weight \(w_k\) and other shape information to be described in §4.

When no confusion arises we use index \(t\) to define both an auto-carrier and the corresponding auto-carrier type, and index \(k\) to denote a vehicle and the corresponding vehicle model.

Since the problem we address has the usual complexity of the practical routing problems tackled nowadays, the concept of a route is also quite involved. In the following we use the triplet \((R, S, t)\) to define a route:

- \(R \subseteq N\) is the sequence of dealers to be visited along the route.
- \(S_i \subseteq M_i\) is the (sub)set of vehicles to be delivered to dealer \(i \in R\) with this route, and \(S = S_1 \cup \cdots \cup S_{|R|}\) is the complete set of vehicles to be delivered.
- \(t\) is the type of auto-carrier used in this route.

In practice, in a route \((R, S, t)\) an auto-carrier of type \(t\), loaded with the vehicles in \(S\), starts from the depot, delivers the vehicles to the dealers in the order specified by \(R\), and then returns empty to the depot. The cost of a route is given by the sum of the costs of the edges traversed by the route. In the following we also make use of a function \(\rho: S \to \{1, \ldots, |R|\}\) that gives the order in which a vehicle is delivered along the route. All vehicles \(k\) \((k \in S)\) demanded by the first dealer in \(R\) have \(\rho(k) = 1\), those demanded by the second dealer in \(R\) have \(\rho(k) = 2\), and so on.

A route \((R, S, t)\) is said to be load feasible if the following conditions apply.

(i) **Weight constraint:** The sum of the weights of the vehicles in \(S\) does not exceed the weight capacity \(W_t\) of auto-carrier \(t\).

(ii) **Two-dimensional loading constraint:** There exists a feasible loading of the vehicles in \(S\) on the \(P_t\) platforms of auto-carrier \(t\): vehicles do not overlap, are completely supported by the loading platforms (possibly translated/rotated/extended), and are completely contained within the maximum cargo length and height specified by the regulation (complete details are described in §4).

(iii) **LIFO policy:** When visiting dealer \(i \in R\), all vehicles in \(S_i\) can be unloaded directly from the auto-carrier, without moving vehicles directed to dealers to be visited later on along the route.

Checking condition (i) is easy, as it is a standard capacity constraint. The check of conditions (ii) and (iii) is difficult, however, as it involves the solution of the two-dimensional loading problem previously discussed. To keep the paper more compact, we describe the details of the loading problem in §4, where we also present the way in which we model the problem and the algorithms we use to solve such model.

We define our combinatorial problem as follows: The **auto-carrier transportation problem** (A-CTP) calls for the determination of a set of routes of minimum cost, such that each route is load feasible, the demands of the dealers are completely fulfilled, and at most \(K_t\) auto-carriers are used for each auto-carrier type \(t\). The NP-hardness of the A-CTP can be proven with a reduction from the CVRP.

In the following we suppose that the fleet of auto-carriers is large enough to allow the delivery of all the requested vehicles. Moreover, we forbid the splitting of the delivery requested by a dealer inside one single route. Indeed we note that, because of the LIFO requirement, a route might be feasible only if the dealer demand was split into two or more subsets, not contiguously loaded on the auto-carrier. However, we forbid this splitting and allow a route to visit a dealer at most once, because such splitting rarely gives rise to a routing cost reduction and is disliked by dealers and drivers.

### 3. Literature Review

The A-CTP is the combination of two problems: (i) how to load the vehicles into the auto-carrier platforms and (ii) how to route the auto-carriers along the road network. The algorithm we propose is the first optimization algorithm that solves this problem. Indeed, according to our knowledge, the other optimization algorithms proposed in the literature for auto-carrier transportation solve either just the routing (with a very simplified loading policy) or just the loading. In this section we first survey the main results in the related literature and then describe how our algorithm differs from others.

Agbegha (1992), and later Agbegha, Ballou, and Mathur (1998), describe the best practices used by companies to deliver vehicles through auto-carriers in the American market; they then focus their attention on the loading subproblem. They model the auto-carrier as a set of a fixed number of slots and introduce a loading network to describe LIFO precedence
among slots. Each vehicle is assigned to a slot, taking into consideration the loading network and additional pairwise incompatibilities slots/vehicles. The resulting problem is a nonlinear NP-hard assignment problem that they solve with a branch-and-bound technique. They do not propose any routing algorithm.

We could not replicate the results by Agbegha (1992) to model modern European auto-carriers. Apparently the research of the last two decades on loading equipment increased to a large extent the flexibility of these carriers; these now can easily be adapted to transport one or two heavy trucks or up to 12 small cars. As a consequence we could not model them using pairwise incompatibilities and a fixed number of slots (details on our approach are given in §4).

Tadei, Perboli, and Della Croce (2002) study the case of an Italian vehicle transportation company. They consider both the routing and the loading aspects and provide an overall heuristic based on an integer linear programming (ILP) formulation. The problem they address is very complex, as it involves multiple days and maximization of profits (equal to gross revenues minus penalties for late deliveries). Also because of this large complexity, they make use of two relaxations. First, they relax the loading sub-problem. For each auto-carrier they compute an equivalent auto-carrier length that is derived from the total auto-carrier loading platform lengths and a constant taking into account the auto-carrier loading equipment. Then they divide the set of vehicles into different loading classes according to their shape and compute for each vehicle an equivalent vehicle length that is obtained by multiplying the original length by a factor that depends on the loading class. The loading problem is thus modeled as a single capacity constraint: the sum of the equivalent vehicle lengths should not exceed the equivalent auto-carrier length. Second, they relax the routing problem by grouping all possible destinations into clusters. They build an initial solution by forcing each auto-carrier to load only vehicles belonging to the same cluster, and then improve the solution by allowing the auto-carrier to load vehicles from neighbor clusters. They do not output routes but make assignments of auto-carriers to clusters. Because of these two relaxations we cannot use their algorithm to obtain the detailed information on how to load vehicles into platforms and how to route auto-carriers.

Miller (2003) studies both routing and loading aspects of an auto-carrier transportation problem occurring in the U.S. market and develops a greedy heuristic followed by inter- and intraroute local search optimization. To decrease the complexity he introduces some limitations. For the routing part, he does not determine the travel distances between the pairs of locations. For the loading part he models the auto-carrier as two flat loading platforms, removes technical limitations (that would prevent assignment of some vehicles to some platforms), and supposes vehicles are loaded straight on the two platforms. In this way the loading problem becomes a bin packing problem with two bins.

More recently, Cuadrado and Griffin (2009) address a real-world auto-carrier distribution case in Venezuela. They solve the medium-term capacity planning (i.e., the problem of determining a good size for the fleet of auto-carriers) with an ILP model, and address the daily assignment of trips and loads to the transport units with a two-phase heuristic directly derived from that of Tadei, Perboli, and Della Croce (2002).

Jin et al. (2010) compare the transportation of vehicles via road and railway in the U.S. market. They use a business scheme to minimize the total distribution costs and solve it with an ILP formulation. They do not propose any routing algorithm. They relax the routing problem by grouping all possible destinations into clusters. They build an initial solution by forcing each auto-carrier to load only vehicles belonging to the same cluster, and then improve the solution by allowing the auto-carrier to load vehicles from neighbor clusters. They do not output routes but make assignments of auto-carriers to clusters. Because of these two relaxations we cannot use their algorithm to obtain the detailed information on how to load vehicles into platforms and how to route auto-carriers.

Jin et al. (2010) compare the transportation of vehicles via road and railway in the U.S. market. They use a business scheme to minimize the total distribution costs and solve it with an ILP formulation. They do not propose any routing algorithm. They relax the routing problem by grouping all possible destinations into clusters. They build an initial solution by forcing each auto-carrier to load only vehicles belonging to the same cluster, and then improve the solution by allowing the auto-carrier to load vehicles from neighbor clusters. They do not output routes but make assignments of auto-carriers to clusters. Because of these two relaxations we cannot use their algorithm to obtain the detailed information on how to load vehicles into platforms and how to route auto-carriers.
4. Solution of the Loading Problem

In this section we present an algorithm to determine if a given route \((R, S, t)\) is load feasible, according to conditions (i)–(iii) of §2, or not. As mentioned before, the exact solution of the original two-dimensional loading problem is difficult in practice. Thus, we content ourselves with an approximate modeling that we base on the results of Agbegha, Ballou, and Mathur (1998) (reusing their concept of loading network) and the results of Tadei, Perboli, and Della Croce (2002) (keeping their concept of loading classes and equivalent lengths, but extending them to each loading platform of an auto-carrier). The reliability of our approximate modeling was tested together with the logistic company. In the following we denote homogeneous as a loading that involves identical vehicles and heterogeneous as a loading that involves vehicles of different models.

For each vehicle model and each auto-carrier type, the logistics company provided the load index for the most common vehicle models. For each pair of platforms such that \(p \prec q\), our empirical determination of a value \(d_{pq}\) for \(d_{pq} = \sum_{k \in S} w_k > W_i\) then return infeasible
\[ f_i = \sum_{k \in S} \frac{1}{d_{kt}}. \] (1)

If the loading is homogeneous, then all vehicles \(k \in S\) have the same load index \(d_{kt}\). By the definition of load index it follows that at most \(d_{kt}\) such vehicles can be packed in this auto-carrier. Hence, we return feasible if \(f_i \leq 1\) and infeasible otherwise. Just to give an example, for the homogeneous loading of Figure 1 we have \(f_i = \sum_{k \in S} \frac{1}{d_{kt}} = 1\) and the load is feasible. The information obtained by computing \(f_i\) is very useful for homogeneous loadings, but very approximate for heterogeneous loadings. Indeed, a heterogeneous loading may also be infeasible when \(f_i \leq 1\) and feasible when \(f_i > 1\). We empirically determined a value \(f_{\max} = 1.2\)

Whenever \(f_i \leq f_{\max}\) and the loading is heterogeneous, we determine the feasibility of the loading by invoking a combinatorial procedure based on an ILP model. The ILP model is important for a full comprehension of the loading problem we address. To describe the model and the procedure used to solve it we need some additional notation.

To model LIFO constraints we define a set of precedence relations among the loading platforms: \(p \prec q\) if a vehicle loaded on platform \(p\) forbids the unloading of a vehicle loaded on platform \(q\). For example, in Figure 1 a vehicle loaded in the rear bottom platform does not allow vehicles from the top platforms to be loaded or unloaded, as it cannot lower the rear top platform. For each auto-carrier we define a precedence graph having \(P\) vertices and an arc \((p, q)\) for each pair of platforms such that \(p \prec q\). Our precedence graph is related to the loading network used by Agbegha, Ballou, and Mathur (1998), but it refers to loading platforms instead of slots, because modern European carriers are very flexible and cannot be modeled by fixed slots.

The precedence graph of the auto-carrier of Figure 1 is given in Figure 2(a). For sake of clarity the auto-carrier is reported in a smaller dimension in Figure 2(b). The rear lower platform (that we number 1), precedes all other platforms, whereas the rear upper platform precedes only the upper front platform. In more detail, platforms 3 and 4 are lowered during transportation to meet the maximum height limitation imposed by the traffic regulation. When visiting a
dealer, platform 3 is raised, which allows the unloading of any vehicles loaded in platform 1. Then two options are possible: (i) platform 4 is also raised, and in this case vehicles on platform 2 can be unloaded by driving them through the empty platform 1; (ii) the rear part of platform 3 is lowered, which allows vehicles of platform 3 to unload. If the unloading of platform 3 happens before that of platform 2, then the unloading of platform 2 is handled by simply raising platform 3 again. Finally, the vehicles loaded on platform 4 are unloaded by driving them through platform 3 (after lowering its rear part). There is thus no precedence relation between platforms 2 and 3 or between platforms 2 and 4.

Each loading platform $p$ of an auto-carrier of type $t$ has a length $l_{pt}$, a value that also includes the limitations induced by the transportation law. Each platform also has a possible maximum inner extension $A_{pt}$. For example, looking at Figure 1, the front platforms may be extended toward the rear of the auto-carrier by a certain amount, and the rear platforms may be extended toward the front. There is, however, a possible interaction at the center of the auto-carrier between two platforms placed at the lower (resp., upper) level. To model this interaction and limit the total extension of the pair of platforms, given a platform $p$, we use $h(p)$ to denote the platform placed side by side horizontally with $p$. For example, looking at Figure 2, $h(1) = 2$, $h(2) = 1$, $h(3) = 4$, and $h(4) = 3$. The total extension of platforms $p$ and $h(p)$ is limited to be at most $\tilde{A}_{pt} = h(p) + 1$.

To simplify the problem, we considered the different loading configurations that can be used for each auto-carrier type. We grouped vehicle models having the same configuration into a vehicle loading class. In practice, a vehicle loading class includes all vehicle models having similar dimensions and shapes that are loaded in the same way on the auto-carrier type under consideration. This is a reasonable compromise because vehicle models may have very similar shapes. For example, a subset of one of our vehicle loading classes is outlined in Table 1. The last column corresponds to the load index on a typical auto-carrier with four loading platforms, where three vehicles from this class would be loaded on the upper front platform, two on the rear upper platform, two on the lower front platform, and three on the rear lower platform.

In the following we use $c(k, t)$ to denote the class of vehicle $k$ on auto-carrier type $t$.

If vehicles were loaded straight on the platforms, then checking feasibility would have simply consisted of checking that the sum of the vehicles’ length does not exceed the platform length. However, loading devices are available on each platform and allow the loaded vehicles to rotate (by lifting up their front or rears). In this way the vehicle usage of the platform length is reduced, and a larger number of vehicles can be packed.

To model this fact we introduce a length reduction coefficient that depends on the auto-carrier type, the vehicle class, and the platform used. More precisely, let $l_k$ denote the length of vehicle $k$ and $r_{c(k, t), p}$ the length reduction coefficient for a vehicle of class $c(k, t)$ loaded on platform $p$ of auto-carrier $t$. Parameter $r_{c(k, t), p}$ expresses the effect of the rotation on the vehicle length for the given platform. By multiplying the vehicle length by its class reduction coefficient, we obtain

$$\ell_{kpt} = l_k \cdot r_{c(k, t), p},$$

which denotes the equivalent length of vehicle $k$ when loaded on platform $p$ of auto-carrier $t$ in a rotated position.

It is worth noting that our concept of equivalent length generalizes the one used in the work by Tadei, Perboli, and Della Croce (2002). Indeed, the length reduction coefficient they used depends on the vehicle class, whereas ours depends on the auto-carrier type, the vehicle class, and the platform used. The setting of the length reduction coefficient $r_{c(k, t), p}$ has to be particularly accurate. The reduction coefficients are an indirect way to describe this complicated loading problem, and they are useful to solve the real-world

![Figure 2](image-url) Precedence Graph (a) for the Auto-Carrier Type in (b)

Table 1 Some Vehicle Models Belonging to the Same Vehicle Loading Class

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Model</th>
<th>Length</th>
<th>Height</th>
<th>Load index $d_{cl}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevrolet</td>
<td>Matiz</td>
<td>350</td>
<td>151</td>
<td>10</td>
</tr>
<tr>
<td>Suzuki</td>
<td>Alto</td>
<td>350</td>
<td>147</td>
<td>10</td>
</tr>
<tr>
<td>Fiat</td>
<td>Panda</td>
<td>354</td>
<td>154</td>
<td>10</td>
</tr>
<tr>
<td>Fiat</td>
<td>500</td>
<td>355</td>
<td>149</td>
<td>10</td>
</tr>
<tr>
<td>Renault</td>
<td>Twingo</td>
<td>360</td>
<td>147</td>
<td>10</td>
</tr>
<tr>
<td>Daihatsu</td>
<td>Sirion</td>
<td>361</td>
<td>155</td>
<td>10</td>
</tr>
<tr>
<td>Ford</td>
<td>KA</td>
<td>382</td>
<td>151</td>
<td>10</td>
</tr>
</tbody>
</table>
problem only if the approximation introduced is very good. We performed this setting with the help of the logistics company.

Similarly to what was done for \( h(p) \), let \( v(p) \) denote the platform placed vertically above or below platform \( p \) (for example, \( v(1) = 3, v(3) = 1, v(2) = 4 \), and \( v(4) = 2 \) in Figure 2). A vehicle being particularly high when loaded on \( p \) may have an effect on \( v(p) \). For example, we might be forced to completely lower an upper platform, also completely using the length of the lower platform below (see, e.g., Figure 3(a)). Or we might be forced to rotate consistently the upper platform (because of a high vehicle loaded on the upper platform itself or on the corresponding lower platform), losing in this way a portion of the lower platform length (see, e.g., Figure 3(b)).

To express this constraint we define \( \lambda_{k, v(p), t} \) as the equivalent length on platform \( p \) used by vehicle \( k \) when loaded on platform \( v(p) \) of auto-carrier \( t \). The process used to determine \( \lambda_{k, v(p), t} \) is analogous to that used for \( \ell_{k,p} \), but it uses different class reduction coefficients. Also, in this case, the accurate setting has been performed with the help of the logistics company. Just to give a numerical example, we note that in Figure 3(a) the depicted vehicle \( k \) has \( \ell_{k4} = \lambda_{k3} = l_k \) (resp., \( \ell_{k2} = \lambda_{k1} = l_k \)), because it is loaded straight on the upper front (resp., rear) platform and occupies the same space on the lower front (resp., rear) platform.

To model the loading problem as an ILP we define a binary variable \( x_{kp} \) taking value 1 if vehicle \( k \) is assigned to platform \( p \), and value 0 otherwise, for \( k \in S, p = 1, \ldots, P_t \). A continuous variable \( a_p \) is used to define the length extension of platform \( p \), for \( p = 1, \ldots, P_t \). We obtain

\[
\sum_{p=1}^{P_t} x_{kp} = 1 \quad k \in S, \tag{3}
\]

\[
\sum_{k \in S} \left( \ell_{kp} x_{kp} + \lambda_{k, v(p), t} x_{k, v(p)} \right) \leq L_{pt} + a_p \quad p = 1, \ldots, P_t, \tag{4}
\]

\[
x_{kp} + x_{k} \leq 1 \quad p, q = 1, \ldots, P_t; \quad p < q;
\]

\[
0 \leq a_p \leq A_{pt} \quad p = 1, \ldots, P_t, \tag{7}
\]

\[
x_{kp} \in \{0, 1\} \quad p = 1, \ldots, P_t; \quad k \in S. \tag{8}
\]

Constraints (3) impose that each vehicle is loaded on a platform. Constraints (4) model the limit on the maximum length of a platform, also taking into account vertical effects and possible extensions. Constraints (5) impose the LIFO policy. Note that we suppose that vehicles having a different order of visits and being assigned to the same platform can be loaded in such a way that the LIFO policy is satisfied. Constraints (6) model the limit on the maximum extension of two platforms placed side by side (\( p < h(p) \) is used to avoid duplicate constraints), and constraints (7) give the appropriate range to the platform extensions. Procedure check load returns feasible if model (3)–(8) has a feasible solution, infeasible otherwise.

We note that model (3)–(8) defines an NP-hard problem from reduction to the bin packing problem (BPP). Given \( n \) items, each having nonnegative weight \( w_j \) (\( j = 1, \ldots, n \)) and \( m \) bins of identical capacity \( C \), the bin packing problem asks for the minimum number of bins that can accommodate all the items. We can solve the recognition version of a BPP with \( m = P_t \) bins and \( n = |S| \) items by using model (3)–(8) and by setting \( \ell_{kp} = w_j \) and \( \lambda_{k, v(p), t} = 0 \) for all \( k \in S \) and \( p = 1, \ldots, P_t; \quad L_{pt} = C \) for all \( p = 1, \ldots, P_t; \quad p(k) = 1 \) for all \( k \in S \); and \( A_{p, h(p), t} = A_{pt} = 0 \) for all \( p = 1, \ldots, P_t \).
Note, however, that for the auto-carrier loading case we can fix the values of \(|S|\) and \(|P_i|\) (which are integers smaller than or equal to 12 and 4, respectively, in common real-world applications) so the problem has a fixed number of variables and could be solved in polynomial time by the Lenstra (1983) algorithm. However, this fact has only theoretical interest, as we developed algorithms that are very fast in practice.

Indeed, we solved model (3)–(8) through CPLEX 12 and through a specialized algorithm based on an enumeration tree. In this specialized algorithm we first sort vehicles according to their order of visit \(\rho\) (i.e., we first consider the vehicles demanded by the first dealer in the route, then those demanded by the second dealer, and so on). At each level of the tree we create a node by loading the first still unloaded vehicle in any platform, thus obtaining \(O(P_i|S|)\) nodes to be evaluated. The tree is explored in a depth-first fashion.

We implemented three versions of our enumeration algorithm, using different techniques to fathom nodes. The first version we tested is simply a complete enumeration. This is not as bad as it appears, because the number of vehicles and loading platforms is small (\(n \leq 12\) and \(P_i \leq 4\) in our instances).

The second version adopts a fathoming criterion, criterion 1, based on an aggregate continuous relaxation. For any platform \(p\) we remember the available platform length, say, \(a_{p}\), which is the length of the platform that can still be used for loading vehicles.

The value of \(a_{p}\) can be computed as follows. At the root node it is initially estimated by setting it to the maximum length plus maximum extension (i.e., \(a_{p} = L_{pt} + A_{pt}\)) for all platforms. When a vehicle \(k\) is loaded on a platform \(p\), \(a_{p}\) is first decreased by \(\ell_{kp}\) and \(a_{\pi(p)}\) by \(\lambda_{k,\pi(p)}\). Then we try to further reduce \(a_{p}\) by checking in \(O(|S|)\) time if any still unloaded vehicle can be loaded on \(p\), i.e., if there exists a \(k'\) such that \(\ell_{k'p} \leq a_{p}\) or \(\lambda_{k',\pi(p)} \leq a_{p}\). If this condition is never verified, then the available length on platform \(p\) cannot be used to accommodate any vehicle, so we set \(a_{p} = 0\). This simple check is also performed for platform \(\pi(p)\) (i.e., we also set \(a_{\pi(p)} = 0\) in case \(\pi(p)\) cannot accommodate any vehicle). Consider, for example, the loading of Figure 3(a). After loading the first vehicle on platform 4, we set \(a_4 = a_2 = 0\) because no other vehicle can enter platforms 2 and 4, and this reduces the size of the enumeration tree.

At the root node we obtain a lower bound \(t_l\) on the total length of the vehicles to be loaded by computing \(t_l = \sum_{k \in S} \ell_{kp}\), where \(\ell_{kp} = \min_{\pi=p} \cdots p_{k} (\ell_{kp} + \lambda_{kp})\) for \(k \in S\). At level \(k\) we decrease \(t_l\) by \(\ell_{kp}\). A node is fathomed if \(\sum_{q \in P} a_{p} < t_l\). This fathoming criterion provides a rough relaxation of the problem, but it is very quick in practice and gives good computational results (details will be given in §6).

The third version is based on an additional, more enhanced but more time-consuming fathoming criterion, called criterion 2, for any platform \(p\) and any dealer \(i\) we keep in mind the available platform length for the dealer, say, \(a_{ip}\). This is computed as \(a_{p}\) but takes into consideration the LIFO requirement by performing an additional check: If we packed a vehicle \(k'\) in a platform \(q\) preceded by \(p\) (i.e., \(p < q\)), and we have that \(\rho(k')\) is smaller than the position of dealer \(i\) in the route, then we cannot use \(p\) to load vehicles demanded by dealer \(i\); hence, we set \(a_{ip} = 0\). Similarly, we define \(t_i\) as the total platform length needed by any dealer \(i\) by computing \(t_i = \sum_{k \in S} \min_{\pi=p} (\ell_{kp} + \lambda_{kp})\), where \(S_i \subseteq S\) is the set of unloaded vehicles to be delivered to dealer \(i\). We kill the node if \(\sum_{p=1}^{P_i} a_{ip} < t_i\) for any dealer \(i\). Criterion 2 requires an additional \(O(P_i|R|)\) computational effort with respect to criterion 1.

In the third version we also make use of criterion 1 for any node that was not fathomed by criterion 2, because this requires practically no additional computational effort. Indeed, after the computation of criterion 2 we know all the \(a_{ip}\) and \(t_i\) values; hence, we can easily compute \(a_{p} = \sum_{i} a_{ip}\) and \(t_l = \sum_i t_i\) and then fathom the node if \(\sum_{p=1}^{P} a_{p} < t_l\).

5. An Iterated Local Search Approach

To solve the A-CTP we developed an iterated local search (ILS) that follows the footsteps, outlined, e.g., by Lourenço, Martin, and Stützle (2003). Its pseudocode is shown in Algorithm 2, where \(s\) denotes a solution to the A-CTP and \(c(s)\) the solution cost. We start by computing a greedy heuristic solution. Then we enter a loop in which at each iteration we perturb the incumbent solution and apply our local search operators on the modified solution. The loop is performed until a given time limit is reached. We describe the details of the three main components of this ILS (greedy perturbation, local search) in the following sections.

Algorithm 2 (Procedure ILS)

**Input:** An A-CTP instance \(I\)

**Output:** The best solution found \(s_{\text{best}}\)

\(s_{\text{best}} = \text{Greedy Heuristic}(I)\)

Repeat

\(s = \text{Perturbation Method}(s_{\text{best}});\)

\(s' = \text{Local Search}(s);\)

if \((c(s') < c(s_{\text{best}}))\) then \(s_{\text{best}} = s'\)

until time limit;

return \(s_{\text{best}}\).
5.1. Greedy Heuristic

We use a randomized nearest neighbor heuristic based on the execution of two main steps. The first step is deterministic and is aimed at serving large demands. We sort auto-carriers by nonincreasing load capacity. Let \( t \) be the index of the largest auto-carrier. We compute for each dealer \( i \) its fill index on \( t \) as \( f_{it} = \sum_{k=1}^{M_i} \frac{1}{d_{kit}} \). Then, if we find a dealer \( i \) for which \( f_{it} \geq 1.2 \), we load \( t \) with as many vehicles as possible, then route \( t \) to \( i \) and then back to the depot. We then select the next largest auto-carrier \( t \) and repeat the process until \( f_{it} < 1.2 \) for all \( i \in N \). At each iteration we load the vehicles one at a time by nonincreasing order of their load index on \( t \).

In the second step we randomly select an auto-carrier \( t \) among the available ones, and then a dealer \( i \) from among those that still have to be visited. The auto-carrier is chosen with random uniform probability, whereas the dealer is chosen with probability proportional to \( f_{it} \). We load \( t \) with the vehicles in \( M_t \), one at a time by nonincreasing order of load index. If some space is left on \( t \), we look for the dealer nearest to \( i \) whose demand is still not completely satisfied and load its vehicles one at a time on \( t \). We continue by always selecting the dealer nearest to the last visited one. When no more space is left on \( t \), we create the route that starts from the depot, visits the dealers in the order in which they were selected, and then returns to the depot. We then repeat the process with a new route until all vehicles have been loaded.

In both steps a trivial way to check the feasibility of the loading would be to invoke procedure check load of §4 any time we load a vehicle on the auto-carrier. This, however, turned out to be too demanding in terms of computational effort. Hence, we continue extending the route without checking its load feasibility and stop only when the insertion of a new vehicle would make the resulting fill index greater than 1.2. We then check the load, and, if infeasible, we remove the vehicle having the smallest load index from the route. If this is not enough to restore feasibility, we reiterate with the next smallest vehicle until the route becomes feasible. (The removal of the smallest vehicle produced better computational results than the removal of the last loaded vehicle or that leading to the largest cost saving.)

The first step is quite common in heuristic approaches to routing problems with split deliveries. The second step usually does not perform well for instances of the classical CVRP compared to other greedy heuristics; however, it achieved good results in our test because of the particular shape of the road network we consider. Indeed, Italy is long and narrow (see Figure 4), the company depot is located in the north, and if a dealer is visited in the south, then it is better to visit with the same route as the other dealers that are close to it. Other heuristics we attempted, based on different concepts of cheapest insertion (see, e.g., Laporte and Semet 2002) and different levels of randomization, performed worse.

5.2. Local Search Procedures

Eight local search procedures are invoked at each ILS iteration, in the order in which they appear below. Each time a procedure finds an improvement it is re-executed. When it does not provide an improvement, we switch to the following procedure. The process is repeated until all eight procedures fail to provide a further improvement. The split delivery of a dealer is not allowed inside a route; hence, all local search procedures allow a route to visit a dealer at most once. The procedures can be synthetically described as follows.

1. Intraroute move: We select a dealer in its current route \( \langle R^a, S^a, t^a \rangle \) and change its order of visit by removing it from its current position in the route and inserting it into any other position. Loading feasibility may be affected because of the LIFO policy; hence, when the change in the order of visit can lead to a cost reduction, procedure check load is invoked. In total, up to \( |R^a| - 1 \) calls to check load may be needed for each execution of this local search.

2. 1-0 dealer move: We select a dealer \( i \) in its route \( \langle R^i, S^i, t^i \rangle \) and move its subset of vehicles \( S^i_t \) to another route \( \langle R^b, S^b, t^b \rangle \). If \( i \) already belonged to \( \langle R^b, S^b, t^b \rangle \), then we set \( S^i_t = S^i_t \) otherwise we attempt for \( i \) all insertion positions in \( \langle R^b, S^b, t^b \rangle \). The load check is performed on route \( \langle R^b, S^b, t^b \rangle \) any time the move can lead to a cost reduction. Thus, up to \( |R^b| + 1 \) calls to check load are needed.

3. 1-1 dealer swap: We remove a dealer \( i1 \) from its current route \( \langle R^a, S^a, t^a \rangle \) and a dealer \( i2 \) from its current route \( \langle R^b, S^b, t^b \rangle \). We then attempt to insert \( S^i_1 \) in \( \langle R^b, S^b, t^b \rangle \), mimicking what was done in the 1-0 dealer move. If we find a feasible insertion for \( S^i_1 \), then we attempt to insert \( S^i_2 \) in \( \langle R^a, S^a, t^a \rangle \). The loading check is thus performed first on \( \langle R^b, S^b, t^b \rangle \) and then, if the swap is profitable, on \( \langle R^a, S^a, t^a \rangle \); hence, up to \( |R^a| + |R^b| \) calls to check load are needed.

4. 2-1 dealer swap: This is analogous to the 1-1 dealer-swap, but in this case two dealers \( i1 \) and \( i2 \) are removed from \( \langle R^a, S^a, t^a \rangle \) and one dealer \( i3 \) from \( \langle R^b, S^b, t^b \rangle \). The insertions and load checks are performed in a sequential way: First \( S^i_1 \) is loaded, then \( S^i_2 \), and finally, if the swap is profitable, \( S^i_3 \). A maximum number of \( |R^a| + 2|R^b| \) calls to check load are performed.

5. 1-1 model swap: We select a dealer \( i1 \) in route \( \langle R^a, S^a, t^a \rangle \) and a dealer \( i2 \) in route \( \langle R^b, S^b, t^b \rangle \). Then we select a subset \( Q^a_{i1} \) of \( S^a_i \) made by vehicles of
the same model. We then attempt to move $Q_{i1}^t$ from $(R^i, S^t)$ to $(R^j, S^t)$ and $S_{2b}^t$ from $(R^i, S^t)$ to $(R^i, S^j, t^b)$, mimicking what was done in the 1-1 dealer swap. This procedure attempts a smaller change with respect to the 1-1 dealer swap that can be profitable when dealers’ demands are large and difficult to be swapped. Note that if the 1-1 model swap is performed, then the demand of $i1$ is split, but this may be balanced by the possible cost reduction obtained by the change of route for $i2$.

6. 2-1 model swap: This extends 1-1 model swap by selecting $i1$ and $i2$ in $(R^a, S^t)$ and $i3$ in $(R^b, S^t)$ and determining two subsets, $Q_{i1}^t \subseteq S_{1a}^t$ and $Q_{i2}^t \subseteq S_{1b}^t$, each made of vehicles belonging to the same model. It then attempts to swap $Q_{i1}^t$, $Q_{i2}^t$, and $S_{2a}^t$ following the process of the 2-1 dealer swap. Note that the vehicle models used to define $Q_{i1}^t$ and $Q_{i2}^t$ may be different.

7. Auto-carrier interchange: We select a route $(R^a, S^t)$ using an auto-carrier of small dimension and carrying a dealer $i$ with $f_{i,a} \leq 1.2$ (see §5.1 for the computation of this value), but whose demand $M_i$ is currently split into two or more routes. If there is a larger auto-carrier available, say $i^b$, we move $S^a$ into $i^b$. Then we look for the other subsets of $M_i$ that have been split in other routes and try to group them together in $i^b$. To do this, we select the smallest (according to the load index) subset of $M_i$ not already included in $i^b$, remove it from its current route, and load it in $i^b$. We continue with the second smallest subset of $M_i$ and so on until no other subset can be loaded in $i^b$. We then determine the cost reduction, if any. This procedure is aimed at reducing the number of split deliveries without affecting the total number of auto-carriers used. The choice of selecting at each iteration the smallest subset of $M_i$ produced better results than selecting the largest subset. If more than one dealer with $f_{i,a} \leq 1.2$ is served by route $(R^a, S^b, t^a)$, then we attempt the removal of each of them, one at a time, according to their order of visit in the route.

8. Route addition: Let $t^a$ be the index of the largest auto-carrier not used yet. We select a dealer $i$ having $f_{i,a} \leq 1.2$, but whose demand $M_i$ is currently split into two or more routes. We then initialize a route using auto-carrier $t^a$ and remove the smallest (according to the load index) subset of $M_i$ from its current route and load it in $t^a$. We repeat this by removing the second smallest subset of $M_i$, loading it in $t^a$, and invoking procedure check load to determine feasibility. We continue until no other subset of $i$ can be loaded in $t^a$ and determine the new solution cost. To avoid useless computation, as a first step we suppose it may be feasible to load all the split subsets of $M_i$ in $t^a$ and evaluate the new cost. The move is attempted only if there is a possible cost reduction. Procedure route addition increases the number of auto-carriers used but decreases the number of split deliveries. It may lead to cost savings even when auto-carrier interchange failed.

The first four procedures originate from classical algorithms for the CVRP, whereas route addition originates from works on split delivery routing problems and auto-carrier interchange from heterogeneous fleet routing problems. The two model-swap procedures are instead motivated by the particular structure of the customer demands in the A-CTP. After preliminary computational results, we decided to adopt for all procedures a first improvement policy: as soon as an improvement is found, if any, the move is applied and the procedure is restarted. This turned out to be computationally better than best improvement (explore the complete neighborhood and then apply the best move, if any) and other policies based on combinations of first and best improvement. The main reason for this result is that the first improvement policy leads to a smaller number of calls to check load, and this makes each local search algorithm less demanding in terms of computational effort.

An intraroute move is the only optimization involving a single route at a time, whereas all the other procedures are interroute optimizations. The auto-carrier interchange and route addition are explicitly aimed at decreasing the number of split deliveries, whereas procedures 1-0 dealer move, 1-1 dealer swap, and 2-1 dealer swap may decrease the number of splits as a side effect when trying to decrease the routing cost. On the opposite side, procedures 1-1 model swap and 2-1 model swap may increase the number of splits. The LIFO constraint forces a high number of calls to check load, because even the rearrangement of a feasible load on an auto-carrier could lead to a LIFO violation. The decision of keeping together the vehicles belonging to the same model in procedures 1-1 model swap and 2-1 model swap is aimed at keeping the loading as homogeneous as possible, hence facilitating the task of procedure check load.

5.3. Perturbation Method
To perturb the incumbent solution, we destroy a number of routes by
1. selecting with random uniform probability a dealer $i$;
2. determining all the dealers whose “distance” from $i$ is not larger than a given threshold $\gamma$; and
3. removing all routes currently serving the set of dealers determined at the two previous steps.

After this has been done, we restore feasibility by invoking a version of the randomized nearest neighbor (§5.1) that creates new routes by assigning the demands of the removed dealers to the available auto-carriers.

To evaluate the “distance” between a pair of dealers $i$ and $j$, we used the distance on the earth sphere
using the Haversine formula (see Sinnott 1984) on
dealer coordinates. This led to better results than
using the routing cost $c_{ij}$ and the Euclidean distance
between the coordinates. The value of $\gamma$ was empir-
ically set to 200 kilometers after preliminary computa-
tional tests. Similar or more complex perturbation
methods have been applied with success in several
vehicle routing problems (see, e.g., Toth and Vigo
2002), although we could not find any that use the
Haversine distance.

We also attempted other perturbation schemes. We
attempted perturbing the current solution instead of
the incumbent one, but this provided worse compu-
tational results. We then focused on the removal of
routes involving dealers whose demand was split,
but this was somehow redundant with respect to
local search procedures route addition and auto-carrier
interchange. We also attempted the idea of choosing,
at step 2, the set of $p$ dealers being closest to $i$, with $p$
being an input parameter. We attempted several val-
ues for $p$ but always obtained poor results. This may
be explained by the fact that dealers are spread out
over the Italian territory in a very un-uniform way
(see Figure 4), and using a single value of $p$ can pro-
duce much different effects, being destructive when $i$
is located in a sparse area and not destructive enough
when $i$ is located in a dense area. Choosing instead
a fixed distance, independently from the number of
dealers inside such distance, provided a more robust
computational choice.

6. Computational Results
We coded our algorithms in C++ and ran them on a
Pentium Dual-Core, with 2.70 GHz and 2 GB RAM,
running under Windows XP. We tested the algorithms
on instances derived from the real-world problem
faced by the logistics company. We considered the
daily distributions from the main depot of the com-
pany operated in July 2009, obtaining 23 instances,
one for each working day. The fleet we considered
was originally made by four auto-carrier types, but
we decided to keep only the first two types because
they perform 98% of the deliveries. The first type
has four loading platforms and a weight capacity
of 15.1 tons; the second has just two loading plat-
forms and a weight capacity of 6 tons. We have in
total 723 vehicle types, divided in 14 vehicle loading
classes for auto-carrier type 1 and eight vehicle load-
ing classes for auto-carrier type 2. We filled the cost
matrix $c_{ij}$ by computing the shortest distances, in kilo-
meters, between any pair $i, j$ of vertices using GIS-
based software. An example of one of our instances
(09-Jul-01) is depicted in Figure 4. The black square in
the north-center part of Italy denotes the location of
the depot and the circles denote those of the dealers.

To encourage future research on this important
topic, we made the instances publicly available at
www.or.unimore.it/A-CTP. For privacy reasons we
removed from the publicly available files any sensi-
tive information, e.g., the names of the auto-carrier
types and of the vehicle models.

6.1. Comparison with In-Practice
Industrial Solutions
It is difficult to obtain a fair comparison with the
industrial solutions executed by the company, because
many disruptions may occur in every day activity.
Thus, to obtain a fair evaluation of our ILS, we imple-
mented the heuristic algorithm that is used every day
by the company to design the initial daily plan, and
tested it on available instances. In this way we could
compare solutions that referred to the beginning of
the working activity and were independent of possi-
bile successive disruptions. For this comparison we
used the best ILS configuration, which uses the enu-
meration tree with fathoming criterion 1, and stopped
it after 1,500 CPU seconds.

For loading, the algorithm used by the logistics
company supposes that loading is feasible if the fill
load is smaller than or equal to 1.1 and is infeasi-
ble otherwise (whereas we perform an accurate check
through check load). For routing, the algorithm used by
the company executes a clustering of the dealers into
areas, then invokes a constructive greedy heuristic.
that sequentially creates routes by routing an auto-carrier to an area and visiting the dealers in that area. If the auto-carrier is not filled completely after visiting the area, then the route continues through a neighborhood area. The cost of a route is computed only after the route has been built, as the complete matrix $c_{ij}$ is not stored in memory. As usual in real-world transportation problems, a lot of manual work is needed to adjust the routes produced by the software.

A visual representation of the results we obtained is shown in Figure 5, where, for each instance, we provide the solution value provided by the algorithm used by the logistics company, the value we obtained with our ILS, and the intermediate value we obtained using the simplified loading check currently adopted by the company (i.e., the one that supposes that loading on an auto-carrier $t$ is feasible if $f_t \leq 1.1$, infeasible otherwise).

It turns out that the ILS always outperforms the algorithm used by the company, achieving an average improvement of about 10.5%. The ILS with simplified loading also outperforms the algorithm used by the company in all instances, but in this case the improvement averages 5.6%. Hence it appears that the two components of the problem (routing and loading) roughly have a similar impact on the improvements we obtained. We can conclude that the ILS is a better methodology for producing initial daily plans than the algorithm that the company currently uses.

If a disruption occurs during the everyday activity, the company may be interested in rerunning the algorithm to obtain a modified delivery plan, possibly in a short computational time. The ILS we propose is particularly suitable for this purpose, because it quickly converges to good solution values in the first minutes of computation. This fact is stressed by Figure 6,

Figure 5  Comparison of the Performances of ILS, ILS with Simplified Loading, and Algorithm Used by the Logistics Company

*Note.* The vertical axis corresponds to the solution value.

Figure 6  Evolution of the Percentage Gap with Respect to the Best Known Solution with an Eight-Hour Time Limit

*Note.* Vertical axis gives average gap over the 23 instances, horizontal axis gives the CPU minutes.
which we obtained after running the ILS with an 8-hour time limit on each instance. We first evaluated
the percentage gap between the incumbent solution value in a given CPU minute and the solution value
returned at the end of the execution. Then we calculated the average gap per minute over the 23 instances
and reported it on the vertical axis of Figure 6. By running the ILS for five minutes we obtain solutions that
already achieve about 50% of the total improvement. By running it for 25 minutes the solutions achieve
about 75% of the total improvement. After three CPU
hours the curve is almost flat.

6.2. Evaluation of the ILS Performance
We ran the ILS using the enumeration tree with fath-
oming criterion 1 inside procedure check load (see §4),
and stopped it after 1,500 CPU seconds. This time
limit turned out to be a good compromise between
solution quality and the need for a fast execution
(some minutes) for the real-world application. The
results we obtained are reported in Table 2.

To gain some insight in the performance of the
ILS, we evaluated the two main components of the
algorithm—the perturbation method and the local search. The ILS belongs to the large family of
multistart-based metaheuristics (see Lourenço, Mar-
tin, and Stützle 2010). For this family it is usual (see
again Lourenço, Martin, and Stützle 2010) to compare
the perturbation method with the most naïve imple-
mentation obtained by the standard multistart (MS)
algorithm (sometimes also referred to as random restart algorithm). The MS operates as the ILS, with the only
exception that, when trapped in a local minimum,
instead of perturbing the incumbent solution it gen-
erates a new solution by invoking from scratch the
randomized nearest neighbor heuristic (§5.1). Also the
MS was stopped after 1,500 CPU seconds.

In the left part of Table 2, we give the name of the
instance, the number of dealers (column n), and
the total number of vehicles to be delivered (col-
umn M). The smallest instance has 96 dealer requests
and 272 vehicles; the largest one requires the delivery
of 1,139 vehicles using 117 auto-carriers. On average
about 200 dealers are served per day. For both algo-
rithms we provide the number of auto-carriers used
in the best solution found (column nsc), the number
of visits to dealers (nvis), the value of the best solution
(z, in kilometers), the CPU seconds (resp., number
of iterations), elapsed when the best solution was
found (secr, resp., iter), and the total number of itera-
tions (itot). For the MS we also provide the percentage
gap from the ILS, computed as \(\%\text{gap} = 100(z(\text{MS}) - z(\text{ILS}))/z(\text{ILS})\). The best solution values are reported
in bold. The last two lines of Table 2 give average and
total values.

### Table 2: Computational Results of the ILS and Comparison with a Multistart Approach

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>M</th>
<th>nsc</th>
<th>nvis</th>
<th>z</th>
<th>secr</th>
<th>iter</th>
<th>itot</th>
<th>%gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>09-Jul-01</td>
<td>228</td>
<td>832</td>
<td>88</td>
<td>256</td>
<td>51,159</td>
<td>1,254.0</td>
<td>1,432</td>
<td>1,718</td>
<td>89</td>
</tr>
<tr>
<td>09-Jul-02</td>
<td>221</td>
<td>1,139</td>
<td>117</td>
<td>295</td>
<td>55,255</td>
<td>1,425.5</td>
<td>1,104</td>
<td>1,161</td>
<td>117</td>
</tr>
<tr>
<td>09-Jul-03</td>
<td>195</td>
<td>737</td>
<td>81</td>
<td>240</td>
<td>41,490</td>
<td>1,484.9</td>
<td>2,358</td>
<td>2,380</td>
<td>83</td>
</tr>
<tr>
<td>09-Jul-05</td>
<td>243</td>
<td>1,063</td>
<td>110</td>
<td>292</td>
<td>58,526</td>
<td>1,449.1</td>
<td>691</td>
<td>722</td>
<td>109</td>
</tr>
<tr>
<td>09-Jul-07</td>
<td>162</td>
<td>629</td>
<td>66</td>
<td>186</td>
<td>28,739</td>
<td>1,165.3</td>
<td>2,369</td>
<td>3,045</td>
<td>67</td>
</tr>
<tr>
<td>09-Jul-08</td>
<td>206</td>
<td>810</td>
<td>84</td>
<td>231</td>
<td>44,619</td>
<td>1,446.9</td>
<td>1,654</td>
<td>1,713</td>
<td>85</td>
</tr>
<tr>
<td>09-Jul-09</td>
<td>201</td>
<td>941</td>
<td>101</td>
<td>266</td>
<td>51,360</td>
<td>1,593.3</td>
<td>539</td>
<td>1,365</td>
<td>101</td>
</tr>
<tr>
<td>09-Jul-10</td>
<td>199</td>
<td>803</td>
<td>85</td>
<td>240</td>
<td>42,576</td>
<td>981.0</td>
<td>912</td>
<td>1,365</td>
<td>86</td>
</tr>
<tr>
<td>09-Jul-11</td>
<td>244</td>
<td>1,039</td>
<td>108</td>
<td>297</td>
<td>58,458</td>
<td>1,336.9</td>
<td>936</td>
<td>1,045</td>
<td>108</td>
</tr>
<tr>
<td>09-Jul-12</td>
<td>227</td>
<td>826</td>
<td>89</td>
<td>265</td>
<td>44,311</td>
<td>1,183.9</td>
<td>1,315</td>
<td>1,623</td>
<td>90</td>
</tr>
<tr>
<td>09-Jul-13</td>
<td>211</td>
<td>729</td>
<td>79</td>
<td>243</td>
<td>44,439</td>
<td>1,043.3</td>
<td>1,677</td>
<td>2,423</td>
<td>78</td>
</tr>
<tr>
<td>09-Jul-14</td>
<td>206</td>
<td>833</td>
<td>88</td>
<td>244</td>
<td>46,393</td>
<td>1,335.5</td>
<td>1,988</td>
<td>2,234</td>
<td>89</td>
</tr>
<tr>
<td>09-Jul-15</td>
<td>200</td>
<td>801</td>
<td>91</td>
<td>245</td>
<td>47,100</td>
<td>1,393.9</td>
<td>880</td>
<td>942</td>
<td>91</td>
</tr>
<tr>
<td>09-Jul-16</td>
<td>198</td>
<td>707</td>
<td>76</td>
<td>233</td>
<td>34,087</td>
<td>1,173.3</td>
<td>1,637</td>
<td>2,118</td>
<td>76</td>
</tr>
<tr>
<td>09-Jul-17</td>
<td>209</td>
<td>940</td>
<td>105</td>
<td>264</td>
<td>63,090</td>
<td>1,374.3</td>
<td>879</td>
<td>968</td>
<td>106</td>
</tr>
<tr>
<td>09-Jul-18</td>
<td>189</td>
<td>614</td>
<td>66</td>
<td>214</td>
<td>37,336</td>
<td>926.8</td>
<td>1,962</td>
<td>3,194</td>
<td>66</td>
</tr>
<tr>
<td>09-Jul-19</td>
<td>251</td>
<td>875</td>
<td>92</td>
<td>294</td>
<td>51,641</td>
<td>1,215.1</td>
<td>1,191</td>
<td>1,464</td>
<td>93</td>
</tr>
<tr>
<td>09-Jul-20</td>
<td>198</td>
<td>811</td>
<td>87</td>
<td>242</td>
<td>45,360</td>
<td>426.6</td>
<td>392</td>
<td>1,303</td>
<td>88</td>
</tr>
<tr>
<td>09-Jul-21</td>
<td>176</td>
<td>556</td>
<td>59</td>
<td>197</td>
<td>26,997</td>
<td>1,199.4</td>
<td>3,063</td>
<td>3,857</td>
<td>59</td>
</tr>
<tr>
<td>09-Jul-22</td>
<td>221</td>
<td>690</td>
<td>75</td>
<td>257</td>
<td>38,733</td>
<td>1,488.7</td>
<td>2,529</td>
<td>2,546</td>
<td>76</td>
</tr>
<tr>
<td>09-Jul-23</td>
<td>204</td>
<td>614</td>
<td>65</td>
<td>231</td>
<td>35,775</td>
<td>1,379.3</td>
<td>2,911</td>
<td>3,162</td>
<td>66</td>
</tr>
<tr>
<td>09-Jul-24</td>
<td>96</td>
<td>272</td>
<td>30</td>
<td>104</td>
<td>21,054</td>
<td>162.1</td>
<td>2,731</td>
<td>25,688</td>
<td>30</td>
</tr>
<tr>
<td>Avg.</td>
<td>202.1</td>
<td>774.1</td>
<td>82.6</td>
<td>240.6</td>
<td>43,670.6</td>
<td>1,156.9</td>
<td>1,689.0</td>
<td>3,086.0</td>
<td>83.2</td>
</tr>
<tr>
<td>Total</td>
<td>1,900</td>
<td>5,533</td>
<td>583</td>
<td>2,406</td>
<td>44,064.1</td>
<td>663.1</td>
<td>774.5</td>
<td>2,003.2</td>
<td>90.9</td>
</tr>
</tbody>
</table>

Downloaded from informs.org by \[122.108.248.97\] on 17 March 2014, at 01:48. For personal use only, all rights reserved.
For all 23 instances, the ILS finds better solution values than the MS, obtaining a saving of 400 kilometers on average and more than 700 in the best case (instance 09-Jul-09). This clearly demonstrates that in this set of instances the ILS is more effective in searching the solution space than the MS. The fact that the solution is only partially destroyed in the ILS, instead of being completely rebuilt as in the MS, allows the ILS to perform more iterations: 3,086 versus 2,003 on average, with an increase of about 50%. The average sec_2 value produced by the MS is almost half that of the ILS, because the MS cannot improve the solutions found during the first iterations, whereas the ILS continues finding improvements (see Figure 6). It is worth noting that the solutions provided by the ILS use in total 13 auto-carriers fewer than those by the MS but have 41 more visits. This means that the ILS uses more split deliveries to find more efficient loadings. The number of split deliveries can be evaluated by computing (n_vis − n) and is on average close to 38 for the ILS and 37 for the MS.

The routes are pretty small for what concerns the number of dealers: on average, there are three visits per route (but there are two routes, out of the 1,900 routes used in the best solutions, that visit eight dealers each). The number of vehicles transported is on average 9.4 per auto-carrier, which means that good loadings have been found. The largest load in terms of number of vehicles consists of 11 vehicles, the minimum of just two vehicles. The solutions we provide have been checked by the logistics company, which confirmed their feasibility.

To evaluate the impact of the eight local search algorithms we developed, see §5.2; we ran eight additional ILS configurations, each obtained by removing an additional local search. The results we obtained are given in Table 3. The columns have the same meanings as the ones of Table 2. The lines give average values, over the complete set of 23 instances, obtained by the ILS (first line) or by the ILS running without the specified local search algorithm (successive lines).

We note that each local search algorithm has a positive impact on the performance of the ILS. Indeed, all removals lead to a worsening of the average solution value (see column z). The worst results are obtained when removing 1-0 dealer move or 1-1 dealer swap, for which the average solution value increases to 43,818.2 or 43,781.1, respectively. We also note that other local search operators we implemented, consisting of dealer or model swaps involving four or more dealers and/or three or more routes, were finally disregarded because they had a negative impact on the ILS behavior. Overall, the behavior of the ILS is very robust, because each removal does not deteriorate the quality of the solutions found, and still allows the ILS to obtain good improvements with respect to the initial greedy solutions (whose average value was 44,453.2).

### 6.3. Impact of the Loading Check of Feasibility

The solution of the loading subproblem may have a strong impact on the performance of the ILS. In §4 we described procedure check load and proposed a mathematical model and three algorithms based on the exploration of an enumeration tree, to be invoked inside check load. We report in Table 4 the details of the impact of these algorithms on the performance of the ILS. Besides the details on the instance, the table contains four groups of columns, each reporting the results obtained by running the ILS with the specified loading algorithm. We attempted the following configurations:

- solution of the ILP model (3)–(8) by CPLEX 12;
- complete tree enumeration;
- tree enumeration with fathoming criterion 1;
- tree enumeration with fathoming criteria 1 and 2.

As mentioned in §4, the computation of criterion 2 allows us to compute criterion 1 with practically no additional effort; hence, we did not test the tree enumeration with just criterion 2. For each configuration, Table 4 gives the best solution value obtained (z), the number of iterations elapsed (it_{el}), and the percentage of the total CPU time that was spent inside the loading procedure (%load). The best solution value for each instance is reported in bold.

It is evident that the solution of the ILP model is not a good choice, because it consumes on average 98% of the overall CPU time. This causes the algorithm to perform a very small number of iterations. The three algorithms based on the enumeration tree have a much better performance, and the enumeration based on criterion 1 (the one we finally adopted in our ILS) is superior to the other two options: it always finds the best solution value, and it consumes on average the smallest percentage of the total CPU time.

It is also clear that using fathoming criterion 2 inside the enumeration is counterproductive, as it is
too time consuming. Indeed, the fact that $\%load$ is higher for the tree enumeration with fathoming criteria 1 and 2 than for the complete enumeration means that the use of the two criteria is not worthwhile, because they consume too much time without fathoming enough nodes of the tree.

To give an idea of the speed of procedure check load with tree enumeration and fathoming criterion 1 on these instances, we note that on average the procedure is executed 3.14 million times per instance and consumes 534 CPU seconds; hence, each call requires about 2 $\cdot 10^{-4}$ seconds.

6.4. Impact of the Loading Constraints

As noted in §3 the A-CTP belongs to the class of integrated loading and routing problems. Research on this area is motivated by the fact that the presence of loading constraints may influence to a large extent the solution of a transportation problem and thus makes the use of a pure routing model not sufficient. Indeed, if the influence of the loading is small, then one could use an approximated approach taking care only of the routing (as happens for the CVRP, where the loading is relaxed to a simple capacity constraint). If it is large, then integrated loading and routing techniques need to be used. The research has consequently paid attention to the evaluation of the impact of the loading constraints (see, e.g., Gendreau et al. 2006; Fuellerer et al. 2009; Zachariadis, Tarantilis, and Kiranoudis 2012). This evaluation is also important for those approaches that consider loading not as a constraint, but as a penalization factor in the objective function, see, e.g., Ergodan et al. (2012).

In our case, evaluating the impact of the loading means evaluating the impact of conditions (i)–(iii) (see §2), on the difficulty of the problem. To this aim we performed three additional tests. In each test we ran the ILS but relaxed one or more loading conditions. The results we obtained (that can be clearly infeasible with respect to the A-CTP) are shown in Table 5. The standard ILS results, already presented in Table 2, are used for comparison’s sake. The other groups of columns refer, in order, to the tests in which we relaxed condition (i) (No weight), condition (iii) (No LIFO) and conditions (ii) and (iii) (Relaxed loading). When conditions (ii) and (iii) are relaxed all heterogeneous loadings are considered feasible if their fill index is less than or equal to 1.2.

For each group of columns we report the variation with respect to the original ILS run in terms of difference in the number of auto-carryers used ($\Delta n_{acr}$), difference in the number of visits performed ($\Delta n_{vis}$), and percentage gap from $z$ ($\%gap$). We note that the weight constraint has practically no impact on the problem. On average it leaves untouched the values of $\Delta n_{acr}$, $\Delta n_{vis}$ and $\%gap$. Indeed, just a few vehicles are characterized by a very high weight, and in almost all cases the shapes of the vehicles impose the real loading restriction. This consideration is also confirmed by

<table>
<thead>
<tr>
<th>Instance</th>
<th>CPLEX 12</th>
<th>Complete enumeration</th>
<th>Criterion 1</th>
<th>Criterion 1 and 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>$n$</td>
<td>$M$</td>
<td>$z$</td>
<td>$z$</td>
</tr>
<tr>
<td>09-Jul-01 228 832 51,598 18 99.2</td>
<td>51,159 1,618 30.8</td>
<td>51,159 1,718 26.2</td>
<td>51,170 1,393 40.0</td>
<td></td>
</tr>
<tr>
<td>09-Jul-02 221 1,139 65,672 32 97.7</td>
<td>65,224 1,106 28.8</td>
<td>65,248 964 36.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-03 195 737 42,179 53 98.0</td>
<td>41,491 2,318 25.3</td>
<td>41,491 1,980 36.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-04 243 1,063 59,134 26 98.0</td>
<td>58,613 682 51.7</td>
<td>58,467 527 62.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-07 165 629 28,972 53 98.6</td>
<td>28,739 3,003 29.4</td>
<td>28,739 2,403 43.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-08 206 810 45,218 39 98.3</td>
<td>44,622 1,533 44.4</td>
<td>44,693 1,284 53.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-09 200 941 52,013 34 98.5</td>
<td>51,360 1,098 54.9</td>
<td>51,360 1,025 57.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-10 199 803 42,917 50 98.3</td>
<td>42,576 1,443 55.8</td>
<td>42,576 1,365 58.3</td>
<td>42,642 875 73.3</td>
<td></td>
</tr>
<tr>
<td>09-Jul-13 244 1,030 59,100 34 97.7</td>
<td>58,485 912 42.8</td>
<td>58,458 1,045 34.3</td>
<td>58,485 870 45.4</td>
<td></td>
</tr>
<tr>
<td>09-Jul-14 227 826 44,892 41 98.0</td>
<td>44,311 1,567 34.5</td>
<td>44,311 1,306 45.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-15 211 729 44,678 41 98.6</td>
<td>44,439 2,349 27.7</td>
<td>44,439 1,967 38.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-16 206 833 46,836 55 98.0</td>
<td>46,393 2,129 27.0</td>
<td>46,393 2,123 27.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-17 200 801 47,664 47 98.5</td>
<td>47,108 859 75.1</td>
<td>47,100 942 72.7</td>
<td>47,108 775 77.5</td>
<td></td>
</tr>
<tr>
<td>09-Jul-19 198 707 34,681 31 98.9</td>
<td>34,087 1,988 37.9</td>
<td>34,087 1,188 34.0</td>
<td>34,262 1,595 50.2</td>
<td></td>
</tr>
<tr>
<td>09-Jul-21 209 940 63,446 29 98.7</td>
<td>63,094 657 73.7</td>
<td>63,090 610 70.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-22 189 614 37,963 59 98.4</td>
<td>37,336 2,903 34.8</td>
<td>37,336 2,544 34.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-23 251 875 52,473 40 97.8</td>
<td>51,647 1,266 38.1</td>
<td>51,647 1,179 42.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-24 198 811 45,524 30 98.0</td>
<td>45,360 901 71.7</td>
<td>45,360 1,074 65.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-27 162 552 26,343 102 98.3</td>
<td>25,971 4,821 26.1</td>
<td>25,971 3,750 42.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-28 176 556 27,322 58 98.7</td>
<td>26,997 3,787 26.3</td>
<td>26,997 3,079 39.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-29 221 690 39,461 50 98.2</td>
<td>38,733 2,572 14.8</td>
<td>38,733 2,219 26.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-30 204 614 36,214 60 98.4</td>
<td>35,775 3,097 27.1</td>
<td>35,775 2,589 39.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>09-Jul-31 96 272 21,072 346 98.9</td>
<td>21,034 24,530 20.7</td>
<td>21,034 25,868 16.8</td>
<td>21,034 23,153 25.2</td>
<td></td>
</tr>
<tr>
<td>Avg. 202.1 774.1 44,151.1 58.0 98.4</td>
<td>43,676.3 2,920.0 39.0</td>
<td>43,670.6 3,086.0 34.7</td>
<td>43,697.5 2,568.0 47.6</td>
<td></td>
</tr>
</tbody>
</table>
the number of infeasible solutions with respect to the original problem. Indeed, by checking the solutions obtained after removal of condition (i), we found that five were actually infeasible.

Removing the LIFO policy leads to an average cost saving of 0.63% and to the use of one less auto-carrier. In this case, however, the impact is large for what concerns infeasibility, as all solutions obtained after removing this condition are infeasible with respect to the original problem.

A larger impact is obtained by the removal of conditions (ii) and (iii), which leads to a cost reduction of about 1.65% and to the use of two fewer vehicles. Note that in this case the number of visits increases; hence, a larger use of the split deliveries is needed. In terms of infeasibility the impact is huge: all solutions produced by this configuration were found infeasible with respect to the original problem, and, more precisely, 669 routes of 1,858 were infeasible. We do not report the results obtained by completely removing the loading component from the problem, because such removal is too destructive and the solutions obtained are meaningless with respect to the A-CTP. Overall, the relaxation of the loading constraints and the use of existing algorithms from the CVRP literature is an impracticable approach, because it would lead to solutions characterized by large infeasibilities.

6.5. Penalizing Split Deliveries

Split deliveries are important to reduce the routing solutions costs, but dealers may dislike them (visited many times instead of a few times), as well as drivers (forced to a larger number of stops). To evaluate their effect we performed a study in which we penalized each delivery by a penalty parameter \( \delta \). We expressed \( \delta \) in kilometers, as was done for the routing cost, and attempted all values of \( \delta = 10, 20, \ldots, 100 \). The case in which \( \delta = 0 \) corresponds to the original ILS run reported in Table 2. For each value of \( \delta \) we run the ILS on the 23 instances, with the usual time limit of 1,500 seconds. The results we obtained are shown in Figure 7.

On the x axis we report the value of \( \delta \). On the y axis we report an evaluation of the number of auto-carriers used \( (n_{ac}) \), number of visits performed \( (n_{vis}) \), and kilometers traveled \( (km) \). To compute the values in the figure, we first determine for each attempt the average values over the 23 instances; then we normalize these values with respect to the average values obtained with \( \delta = 0 \).

The values of \( n_{ac} \) and \( km \) are characterized by practically an identical behavior. They increase in almost a linear way when \( \delta \) increases, with a very small slope. The value of \( n_{vis} \) decreases from left to right and is consistently affected by the penalty. A good compromise among routing cost, number of vehicles, and
number of split deliveries is obtained with $\delta = 30$: we save on average 16.2 split deliveries for each day, with just 0.2 vehicles and 150 km more. The company indeed considered this value an appropriate balance of number of deliveries and routing cost.

### 6.6. Pool of Certified Loadings

In other combined routing and loading problems, the use of a pool of loadings whose feasibility was already proved/disproved consistently reduced the computational effort of the algorithm and allowed better solutions. Notably, this was the case in Zachariadis, Tarantilis, and Kiranoudis (2012). In other approaches, however, the use of such pool was disregarded as counterproductive (see, e.g., Alba Martinez et al. 2013). We made an attempt in this direction and created a double pool, storing feasible and infeasible loadings. In procedure check load (see Algorithm 1), before invoking the exact algorithm, we check if the loading is already contained in one of the two pools. If yes we use the information retrieved; otherwise we invoke the exact algorithm and then store the result obtained in the pool for the next iteration. This attempt worsened the results obtained by the ILS. The time spent inside the loading checks increased, the number of iterations decreased, and on average the solution cost increased by 0.6%. A possible reason for this behavior lies in the fact that the exact algorithm is very fast and the number of heterogeneous loadings to be kept in memory is huge (723 vehicle types to be loaded into two auto-carrier types, considering all possible vehicle orderings from the LIFO policy). We also note that an interaction with a database of certified feasible/infeasible loadings is a desirable feature for the everyday activity of the logistics company. This slows down the algorithm but allows a process of automatic learning that can be useful in the long term. We did not discuss this issue in the paper, as we preferred to focus on the optimization topic.

### 7. Conclusions and Future Perspectives

We addressed an interesting real-world transportation problem in which cars, trucks, and other vehicles are loaded onto auto-carriers and then routed to dealers. We proposed a heuristic algorithm that makes use of an enumeration tree to load the vehicles and an ILS to route the auto-carriers.

The modeling of the loading problem that we propose is effective in reproducing real-world loadings and can be solved with a very small computational effort. The overall algorithm obtains important and consistent savings with respect to the solutions obtained by the algorithm currently used by the company and to the solutions produced by a standard multistart approach. The best configuration for the algorithm has been carefully assessed. The impact of the different loading constraints has been evaluated, and this allowed us to conclude that the relaxation of the loading constraints and the use of pure CVRP algorithms is an impracticable approach, because it would lead to solutions characterized by large infeasibilities. The methodology that we present refers to the Italian market but can be easily extended to other markets. The benchmark instances we used are publicly available to encourage further research on the subject.

Throughout the paper we supposed that the number of available auto-carriers is large enough to deliver all vehicles requested in the day (see §2). This was motivated by the instances we addressed that were representative of a period of crisis in the automotive market (July 2009). In case the fleet is not large enough...
enough, then one should postpone some deliveries to the following days. Usually these decisions are based on routing cost reductions, penalties for late deliveries, and other parameters. This is typically the decision process encountered in a dynamic multiperiod routing problem, which represents a topic of great interest for future research activity.

Acknowledgments

The authors thank three anonymous referees, whose remarks greatly improved the quality of this paper.

References


