European Journal of Operational Research 257 (2017) 601–613

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

Models and algorithms for reliability-oriented Dial-a-Ride with autonomous electric vehicles

Victor Pimenta[a], Alain Quilliot[a], Hélène Toussaint[a], Daniele Vigo[b,*]

[a] LIMOS CNRS 6158, Labex IMOBS3, Blaise Pascal University, 63000 Clermont-Ferrand, France
[b] DEI "Guglielmo Marconi" – University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy

ARTICLE INFO

Article history:
Received 2 May 2015
Accepted 16 July 2016
Available online 21 July 2016

Keywords:
Dial-a-Ride problem
ILP models
Resource constrained scheduling

ABSTRACT

We consider a static decision model related to the management of a Dial-a-Ride (DAR) system involving small autonomous electrical vehicles in a closed industrial site. Because of the specific features of the system, in this paper we concentrate on its reliability and propose a model that aims at assigning requests to vehicles by minimizing the number of loading/unloading operations. We propose an integer linear programming formulation of such Stop Number Minimization Problem and examine the behavior of some of its variants. Next, we consider and analyze a set covering oriented reformulation of the model. Finally, we propose a Greedy Randomized Adaptive Search Procedure (GRASP) based heuristic approach that implements insertion mechanisms and is well fitted to realistic dynamic contexts. All proposed methods are tested on benchmark instances involving some tens of requests.

1. Introduction

The current trends in mobility management involve the use of flexible reactive systems, which meet mobility demands in a dynamic way by implementing some vehicle-sharing and by interacting with alternative transportation modes. Such systems strive to find their room between fully individual mobility and traditional collective transportation systems. They also aim at bridging the gap between goods and people mobility, and require the use of advanced technologies (see (Luo & Schonfeld, 2007)) such as Internet, web services, mobile communications, and remote tracking and monitoring. Depending on the context, they may work either as *closed systems*, whose access is restricted to users who accept rules related to mobility tracking, pricing and responsibility, or *open systems*, which work on the basis of a free access/free market principle. Among such systems one may mention *Dial-a-Ride*, *Car-Sharing*, *Car-Pooling* (e.g., AUTOLIB), *Ride Sharing* systems (see, e.g., (Bast et al., 2014); (Boerndorfer, Groetschel, Klostermeiner, & Kuttner, 1999), and (Boyaci & Zografos, 2015)) and routing problems with intermediate facilities (see, e.g., (Ghiani, Lagana, Laporte, & Mari, 2010)).

Recent advances in artificial perception and remote control make now possible new generations of autonomous (i.e., without any driver) individual or collective electrical vehicles, such as Cycab and VIPA (the *Individual Autonomous Vehicles* of Ligier S.A., see http://www.ligier.fr/ligier-vipa and Fig. 1), which lead to the design of new mobility services. In the case of VIPA electric cars, experiments are currently undertaken in Clermont-Ferrand, France to assist internal mobility of both people and objects (internal mail, small packages…) inside the industrial site of Michelin. Other applications are considered for the future, involving hospitals and some pedestrian downtown areas.

Such an experimental service works as a specific *Dial-a-Ride* (DAR) system (see (Boerndorfer et al., 1999); (Cordeau & Laporte, 2007); (Cordeau, 2006); (Cordeau, Gendreau, Laporte, Potvin, & Semet, 2002), and (Quilliot & Deleplanque, 2015)). Users call from *smartphones* or from *ad-hoc* communication devices and wait for the vehicles that serve them. But, since related movements are performed inside a closed and restricted area (smaller than a few square kilometers), users' demands must be handled in a very reactive way. In addition, since autonomous vehicles are involved there are very few routing options and vehicles move along predetermined routes with fixed lengths. Thus, routing as well as minimizing individual and total travel times are not a key issue in this system. Instead, what matters is system *reliability*, that concerns the steady flow of the vehicles traffic and which can be improved through a lean scheduling. In other words, scheduling has to smooth as much as possible the trajectories of the vehicles and has to minimize sequences that require complex interactions between the vehicles.

The model that we consider here is typical of this new kind of problems, and may be viewed as an extension of interval graph coloring models (see, e.g., (Golumbic, 2004)). As previously

* Corresponding author. Fax: +39 0512093073.
E-mail address: daniele.vigo@unibo.it (D. Vigo).

http://dx.doi.org/10.1016/j.ejor.2016.07.037
0377-2217/© 2016 Elsevier B.V. All rights reserved.

Laissez-vous transporter vers le futur.
VIPA, véhicule électrique de transport de passagers sans conducteur ni infrastructure
LIGIER
LASMEA
APOJEE
ViaMéca

Fig. 1. An image of a VIPA car from Ligier S.A.

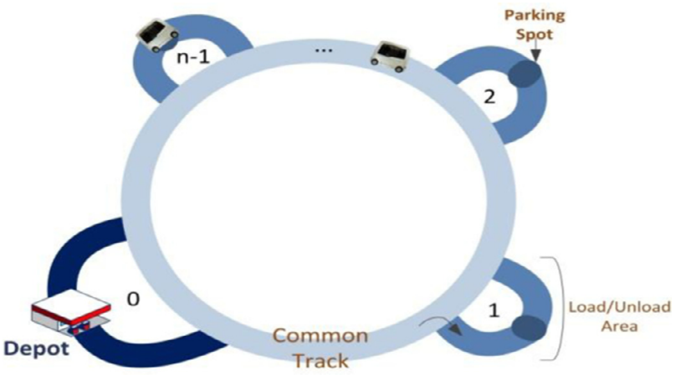

Fig. 2. An illustration of the VIPA circuit.

mentioned, our problem derives from a case study about the management of a specific DAR system, which involves VIPA autonomous vehicles working in real-time as "*horizontal elevators*", moving people or goods in a district or an industrial site. A set of autonomous vehicles travel between several stations located along a circuit serving requests, each having a specific origin and destination station: the main constraints of the problem are those of a classical DAR problem. However, our main focus is on travel time reliability, which is mainly correlated with the number of *stops* performed during the service. Hence, we aim at serving the requests in such a way that the vehicles minimize the number of stops they perform, thus enabling them to stay as much as possible on the main track without deviating to the *loading/unloading* areas. In practical settings, such decision problem has to be solved *on line* and the performance of the solution methods, inevitably of a heuristic nature, has to be evaluated through discrete-event simulation. However, both to better understand the structure of the problem and to get benchmarks for heuristic solutions we study here a static (*off-line*) version of the problem.

The paper is organized as follows. In Section 2, we define the *Stop Number Minimization Problem* (SNMP) in a formal way and introduce an integer linear programming (ILP) formulation. We also discuss variants and extensions of the SNMP, and propose a *set covering* reformulation that avoids considering the number of vehicles as a parameter of the model. Next, in Section 3 we discuss some theoretical properties of our model and perform an experimental analysis of the behavior of its different ILP formulations. Finally, in Section 4, we propose and test a greedy randomized adaptive search procedure (GRASP) heuristic based on an insertion mechanism, which is well-fitted to *on-line* contexts. Finally in Section 5 we draw some conclusions and outline possible future research directions.

2. Problem definition and modeling

In our problem we consider a set of autonomous VIPA vehicles traveling along a closed network that has the shape of a circuit Γ with n nodes, while serving a set of DAR demands. In practical settings also other network topologies can be considered, such as tree or star ones, but in all such cases the common feature is that there is only one elementary path connecting each origin o to a destination d of a transportation *request*. As a consequence, routing decisions are not part of our problem The nodes of the circuit Γ are denoted by 0, …, $n-1$, where node 0 represents the *Depot,* that is the base of the vehicles. The remaining nodes correspond to parking slots that may be either the origin or the destination of a request. The vehicles travel the circuit always in the same direction. Hence, when a request calls for the transportation of a load from an origin o to a destination d, the trajectory of the vehicle is $\{o, (o+1)$ Mod n, $(o+2)$ Mod $n, \ldots, d)$, where Mod is the modulo division operator. Circuit Γ is made up of a *common track* and of *loading/unloading areas*, associated with nodes $\{0, \ldots, n-1\}$ as illustrated by Fig. 2. Note that in our application vehicles have a lateral loading and unloading access. Hence, the circuit has to be traveled counterclockwise to allow loading and unloading be performed at the stops through platforms that are located outside the circuit itself.

The speed of the vehicles on the common track is considered constant, and in our application it is equal to about 15 kilometer/hour. Overtaking is forbidden on the main track and when a vehicle leaves a *loading/unloading* area it has no precedence on the vehicles which travel on the main track. The time required by a vehicle to move between two stops clearly takes into account the decelerations and accelerations required before and after the stop. The time a vehicle spends in a *loading/unloading* area is a random variable, increasing in a weak (sub-linear) way with the number of loading/unloading operations that are performed on this area.

Traveling along the whole circuit without stopping only takes a few minutes. Therefore, a vehicle is allowed to travel several times along Γ (i.e., performing so-called *waiting laps*) before actually serving an user. However, the load of a request may not stay on the vehicle for a full tour, i.e., once loaded it has to be unloaded as soon as the vehicle reaches its destination. A solution is therefore defined by assigning to each request j the vehicle that serves it and the number h of waiting laps, i.e., the number of times the vehicle is going to travel the whole circuit Γ before it starts the service of that request.

We are given a fixed number K of identical vehicles, all located initially (i.e., at time 0) at the *Depot* node. All the vehicles have the same capacity Q, which represents either a maximum weight or a number of passengers that can be transported. We denote by H the largest number of *waiting laps* for the vehicles, which is the maximum number of tours a vehicle is allowed to perform before starting the service of a request. Note that because the numbers of vehicles and of waiting laps are fixed it may happen that some request remains not serviced (i.e., is rejected). However, as our main focus is on travel time reliability we avoid considering the management of rejected requests. As a consequence, we assume that

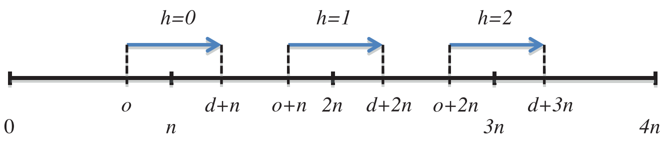

Fig. 3. The Stop Node Set obtained by unfolding a circuit with two waiting laps.

the vehicle number K is large enough to enable the system to meet all requests while respecting the maximum waiting lap restriction.

We adopt a *static* point of view, where all the user requests are known in advance (i.e., at time 0). A *request* (or *demand*) j is defined by a triple ($o(j)$, $d(j)$, $L(j)$) including an *origin node* $o(j)$ and a *destination node* $d(j)$, both in the set $\{0,\ldots, n-1\}$, together with a *load* $L(j)$. This defines the SNMP (*Stop Number Minimization Problem*) problem. In case all loads $L(j)$, $j = 1,\ldots, m$, are equal to 1, the problem is called *Unit* SNMP. Furthermore, we suppose here that users ask for the system only when they are ready to move, therefore a request does not involve any kind of temporal requirement such as time windows.

Since we forbid that the load of a request stays on board of a vehicle during a full tour, we make here the assumption that individual *travel times* of the requests are always constant and need not be optimized as is, instead, done in classical DAR problems. We note here that the number of times a vehicle stops while moving from origin $o(j)$ to destination $d(j)$ to serve other requests has clearly an impact on the duration of the individual *travel time* of j. We will address this issue later, in Section 2.2 where we discuss SNMP variants and *Quality of Service*. However, we also note that, because of the accelerations and decelerations required to stop at a loading/unloading area, minimizing the number of stops will also help minimizing the *global travel time* of the fleet.

2.1. An ILP formulation of the Stop Number Minimization Problem

To simplify the model description let us unfold the circuit Γ into a linear ordered set $I(\Gamma) = \{0,\ldots, (H+2)\cdot n\}$ of nodes, each representing the visit of a node of the circuit during one of the H waiting laps, as depicted in Fig. 3. The set $I(\Gamma) = \{0,\ldots, (H+2)\cdot n\}$, where both 0 and $(H+2)\cdot n$ correspond to the depot, is called the *Stop Node Set*,. Let us also define $p = (H+2)\cdot n - 1$ as the index of the last stop in $I(\Gamma)$.

Let D be the set of all requests $j = (o(j), d(j), L(j))$, $j = 1,\ldots, m$, where if $d(j) < o(j)$, we replace the destination with $d(j) + n$. In this way, we may associate with each request $j = 1,\ldots, m$, a collection of $H+1$ discrete intervals $I_{j,h} = \{o(j) + h\cdot n, \ldots, d(j) + h\cdot n\}$, $h = 0,\ldots, H$, of the *Stop Node Set,* where the service of the request must take place. Without loss of generality, we suppose that each node $i = 1,\ldots, n-1$ is active for our problem, i.e., that there exists at least one index value j such that either $i = o(j)$ or $i = d(j)$. If this is not the case, we may remove all *non-active* nodes from the set $\{1,\ldots, n-1\}$, reduce n and renumber the *active* nodes accordingly.

Furthermore, for each node i in the set $I(\Gamma)$, $i \neq (H+2)\cdot n$, we define the following two sets:

- $Cross(i) = \{(j,h), j = 1,\ldots, m, h = 0,\ldots, H$, such that $o(j) + h\cdot n \leq i < i+1 \leq d(j) + h\cdot n\}$. Hence, $(j,h)\in Cross(i)$ means that a vehicle leaving node i must carry the load $L(j)$ of request j served with waiting lap equal to h;
- $Stop(i) = \{(j,h), j = 1,\ldots, m, h = 0,\ldots, H$, such that $(i = d(j)+ h\cdot n)$ OR $(i = o(j)+ h\cdot n)\}$. Hence, i is either the origin or the destination node of request j served with time lap h.

Then, we get the following ILP model for SNMP, where the number K of available vehicles is fixed to a suitably large number. The model uses the following binary decision variables. Variable $Z_{j,k,h}$ takes value 1 if vehicle $k = 1, \ldots, K$ serves request $j = 1, \ldots, m$ with *waiting lap* $h = 0, \ldots, H$, and 0 otherwise. Variable $T_{k,i}$, takes value 1 if $i = 1, \ldots, p$ is a *stop node* for vehicle $k = 1, \ldots, K$, and 0 otherwise. The model is then:

$$\textbf{SNMP}(\boldsymbol{K}) \quad \min \sum_{k=1}^{K}\sum_{i=0}^{p} T_k i \tag{1}$$

subject to:

$$\sum_{k=1}^{K}\sum_{h=1}^{H} Z_{jkh} = 1, \quad \forall j = 1,\ldots,m \tag{2}$$

$$\sum_{(j,h)\in Cross(i)} L(j).Z_{jkh} \leq Q, \quad \forall k = 1,\ldots, K; \forall i = 0, \ \ldots, \ p \tag{3}$$

$$Z_{jkh} \leq T_{ki}, \quad \forall k = 1,\ldots, K; \forall i = 0, \ \ldots, \ p; \ \forall (j,h) \in Stop(i) \tag{4}$$

$$Z_{jkh} \in \{0,1\}, \quad \forall j = 1,\ldots,m; \forall k = 1,\ldots, K; \forall h = 0, \ \ldots, \ H \tag{5}$$

$$T_{ki} \in \{0,1\} \quad \forall k = 1,\ldots, K; \forall i = 0, \ \ldots, \ p, \tag{6}$$

where the objective function (1) minimizes the number of stops, constraints (2) impose that each request is assigned exactly once. Constraints (3) are the capacity constraints and (4) are the coupling constraints for the decision variables. Finally constraints (5) and (6) define the decision variables. Note that the optimal solution of SNMP(K) uses at most K vehicles, i.e., if convenient it may leave some vehicle unused. Further discussion on the available number of vehicles will be provided in the following sections.

2.2. Extensions and variants of the SNMP

The problem we defined in the previous section can be easily extended and modified to consider further characteristics of real-world problems. In the following we present some of such variants.

First of all, the SNMP model can be easily extended to consider time windows for the service. These are modeled by imposing lower and upper bounds $Min(j)$, $Max(j)$, $j = 1,\ldots, m$ on the values of the *waiting laps* associated with each request. Hence, service time window is imposed by adding to SNMP the following set of constraints:

$$Z_{jkh} = 0 \quad \forall h \notin \{Min(j), \ \ldots \ , \ Max(j)\}; \forall k = 1,\ldots, K. \tag{7}$$

Let us now denote by *Load-Vehicle*(K), LV(K) for short, the feasibility problem which derives from restricting the SNMP with K vehicles to only consider decision variables Z. The resulting problem has no objective function and includes the constraints (2, 3, 5) and (6) of SNMP above. Then by computing the smallest value $K = K_{min}$ such that LV(K) has a feasible solution, we model the variant in which the objective is that of minimizing the number of used vehicles.

Another variant of SNMP, called *Global Travel Time Minimization Problem* (GTTMP), calls for the minimization of the total number of tours performed by the K vehicles to serve all the requests. To this end, let us define a continuous decision variable r_k, $k = 1, \ldots, K$, which is defined as the largest value of waiting laps associated with a request served by vehicle k. Thus, $r_k + 1$ is equal to the total number of complete circuits traveled by vehicle k. Given the value K, the model of GTTMP is

$$\textbf{GTTMP} \quad \min \sum_{k=1}^{K} r_k \tag{8}$$

subject to:

$$\sum_{k=1}^{K}\sum_{h=1}^{H} Z_{jkh} = 1, \quad \forall j = 1,\ldots,m \tag{9}$$

$$\sum_{(j,h)\in Cross(i)} L(j)\cdot Z_{jkh} \le Q, \quad \forall k=1,\ldots, K; \forall i=0, \ldots, p \tag{10}$$

$$h\cdot Z_{jkh} \le r_k \quad \forall k=1,\ldots, K; \forall j=1,\ldots,m; \forall h=1,\ldots,H \tag{11}$$

$$Z_{jkh}\in\{0,1\}, \quad \forall j=1,\ldots,m; \forall k=1,\ldots, K; \forall h=0, \ldots, H \tag{12}$$

$$r_k\ge 0, \quad \forall k=1,\ldots, K, \tag{13}$$

where the objective function (8) minimizes the total number of circuits. Constraints (9) impose that each request is assigned exactly once. Constraints (10) are the capacity constraints and (11) are the coupling constraints for the variables. Finally, constraints (12) and (13) define the decision variables.

2.2.1. Quality of service

Both SNMP and GTTMP reflect the common viewpoint of the fleet managers in the VIPA context, i.e., in the context of vehicles performing services inside an industrial area, where managers pursue a specific industrial purpose and are seeking to minimize the total service time for all requests. Clearly such solutions may be sub-optimal if we consider the viewpoint of users for which the number of stops performed between origin and destinations has certainly an impact on users' *individual travel time*. A first possibility of considering individual travel times consists in setting the *individual waiting time* of user j to the value $h(j)$ of the *waiting lap* associated with j. Then SNMP can be easily extended by including into the objective function the sum of $h(j)$ for j=1, …n. In such a case, the experiments we conducted with this extended objective function show that minimizing the *Stop Number* and minimizing the *Waiting Lap Number* are not convergent criteria. Indeed, minimizing the *Waiting Lap Number* tends to use as many vehicles as possible, while instead, minimizing the *Stop Number* tends to let a given vehicle perform as many laps as possible. In addition, a more precise modeling of individual travel times should explicitly take into account the number of stops performed between origin and destination of the request. It is, however, clear that the corresponding model would be more complex and its consideration goes beyond the scope of the present paper. In the analysis of computational tests performed in Section 3.2 we will briefly discuss the distribution of stops between origin and destinations of requests.

2.3. A set covering reformulation of SNMP

As it also happens for the graph coloring problem (see, e.g., (Cornaz & Jost, 2008); (Golumbic, 2004); (Luo & Schonfeld, 2007); (Malaguti et al., 2011), and (Mehrotra & Trick, 1996)) the linear relaxation of model (1)–(6) can be weak, particularly when the number of available vehicles is large with respect to K_{min}. This behavior is confirmed in our computational testing, discussed in Section 3.3, which shows that by using such formulation it is possible to solve only small-size instances. Therefore, following what is done for similar problems, such as graph coloring (see, e.g., (Cornaz & Jost, 2008)), we introduce a different model based on a set-covering reformulation of SNMP. To this end, let us define the notion of *feasible service* as a pair $s = (J, h)$, where J is a subset of requests, and $h(\cdot)$ a function from J to {0,…, H}, that associates a waiting lap to each $j\in J$ and such that

$$\sum_{j\in J:(j,h(j))\in Cross(i)} L(j)\le Q, \quad \forall i=0,\ldots,p. \tag{14}$$

Clearly, a feasible service identifies a set J of requests which can be served by a vehicle together with, for each such request $j\in J$, its related waiting lap $h(j)$. According to this, we use the notation $j\in s$ to say that request j is served by service s.

We denote by S the set of all feasible services, and for each $s\in S$ we define N_s as the number of stops performed by the vehicle executing s. The set-covering reformulation of SNMP, called SC-SNMP uses binary decision variables X_s, which take value 1 if the feasible service s is selected and executed by a vehicle.

$$\mathbf{SC-SNMP} \quad \min \sum_s N_s\cdot X_s \tag{15}$$

subject to:

$$\sum_{s\ |\ j\in s} X_s = 1, \quad \forall j=1,\ldots,m \tag{16}$$

$$X_s\in\{0,1\}, \quad \forall s\in S, \tag{17}$$

where the objective function (15) minimizes the number of stops. Constraints (16) impose that each request is assigned exactly once and constraints (17) define the decision variables.

We notice that this *Set Covering* oriented reformulation does not involve the parameter K. Also, it is very generic because variants, such as GTTMP, may be cast into the same framework in a very natural way.

3. Preliminary theoretical and experimental analysis

In this section we analyze in detail the SNMP and provide some insights with respect to its complexity and the possibility of solving it to optimality by analyzing the results we obtained on benchmark instances. The heuristic solution of SNMP, which is more appropriate in the *on-line* contexts, will be addressed in Section 4. We also compare experimentally with SNMP some of the variants that were introduced in the previous section.

3.1. Discussion on problem complexity

The problem we introduced in the previous section has clear relations with other important families of optimization problems, such as the interval graph coloring. It is, therefore, important to address the worst-case computational complexity of the problem. To this end, we recall that an instance of the *2-Partition Problem* (2PP) is defined by a collection a_1, …, a_m of positive integer numbers, such that $S = \Sigma_{i=1..m} a_i$ is even, and that solving it means computing a subset A of {1, …, m} such that $\Sigma_{i\in A} a_i = S/2$. Such a problem is known to be NP-Hard according to Garey and Johnson (1979). We then introduce the following:

Theorem 1. *In the case where $H = 0$, and no hypothesis is done about the capacity Q and the loads $L(j)$, $j = 1,\ldots, m$, the SNMP is NP-Hard.*

Proof. Let us check that the 2PP problem may be viewed as a specific case of our *Stop Minimization* problem. To this end, we consider a collection of requests $j = 1,\ldots, m$ such that:

- $o(j) = 1$ and $d(j) = n$-1 for each $j = 1,\ldots, m$;
- $Q=\frac{1}{2}\sum_{j=1}^{m} L(j)$.

In such a case, the optimal value of the related instance of *Stop Minimization* is equal to 4 if and only if the instance of the 2PP which is defined by Q and the loads $L(j)$ has a solution.□

In the remaining part of this section, we are going to perform a brief analysis of different restrictions of *SNMP*, to better understand which features make the problem difficult, and which ones make it close to be polynomial in time. The complexity proof above mainly relies on an argument involving the loads of the requests. So, from the complexity point of view, an interesting special case of SNMP is clearly that with unit load request, i.e., with $L(j)$=1, $j = 1,\ldots, m$, and denoted as *Unit SNMP* (USNMP), Unfortunately, we were not able to derive a tight bound on the complexity of such variant, but we have the following:

Conjecture: *USNMP is NP-Hard, even when $H = 0$.*

However, some remarks may be raised showing that, in the case with H=0, the USNMP is close to being polynomial in time. First of all, we may notice that, if H=0, then the constraint matrix associated with the LV(K) feasibility model (see Section 2.2) is totally unimodular: this may be obtained, through a straightforward calculus about determinants, as a consequence of the fact that, for every $k = 1,\ldots, K$, the matrices M_k defined by the capacity constraints are all interval matrices and are all equal. As a consequence, if Q=1, we see that the feasible solutions of our problem are exactly the feasible solutions of the *vertex coloring* problem defined by the color number K and by the interval graph induced in a natural way by the request set D on the discrete linearly ordered set 0,…, 2n. Furthermore, if $H = 0$, the vertices of the polyhedron defined by the LV(K) feasibility model are all integral, and, solving the problem of minimizing the number of used vehicles in SNMP can be performed in polynomial time. Moreover, if $H = 0$, the unit-load variant of such vehicle-number minimization problem can be solved through a min-cost flow algorithm (see, e.g., (Golumbic, 2004)).

Another interesting case arises when both K and Q are fixed and H=0: here SNMP is polynomial in time even with unrestricted load values $L(j)$, $j = 1,\ldots, m$.

Theorem 2. *If H=0 and if Q, K are fixed, then the SNMP can be solved in polynomial time.*

Proof. We prove that, if $H = 0$ and Q is fixed, then our problem can be viewed as a shortest path problem defined on an acyclic network with polynomial size, which is solvable in polynomial time through dynamic programming.

To this end, let us describe how the network $G = (N, A)$ is constructed. We define a state vector as any vector $F = F_{k,c}$, $k = 1, \ldots, K$, $c = 1, \ldots, Q$, with values in [0,$n-1$], and such that, for any k, $F_{k,Q} \leq \ldots \leq F_{k,1}$. Let F denote the set of all state vectors. As we will see later, we will replicate such state vectors for each node i. Vector F tells us, for any node i, and any vehicle k, that the load of vehicle k is going to be at least equal to c while k is moving from i until $i + F_{k,c}$.

We define the Time Space TS as the set of all pairs $t = (i, j)$, $i = 0, \ldots, n-1$, $j = 1, \ldots, m$, such that $o(j) = i$. The space TS is linearly ordered in a natural way by the lexicographic order: (i, j)≺(i', j') if and only if $i < i$' OR (($i = i$') AND ($j < j$')). For any $t \in TS$, we denote by $\sigma(t)$ its successor in TS according to the linear ordering. In case t is the last element of TS with respect to such ordering, we set $\sigma(t)$ = Nil. Then we set N = F×TS ∪ {End}, where End is a fictitious node, and F×TS denotes the Cartesian product of the State Space F and the Time Space TS. The meaning of a node (F, (i, j)) of N is that, while we are scanning the set 0, …, $n-1$, we assign a vehicle to every request j' such that ($o(j)$,j') ≺ (i,j) and we have to decide which vehicle we assign to j. Then, the coefficient $F_{k,c}$ indicates that vehicle k will carry a load at least equal to c when moving from i to i+ $F_{k,c}$.

In addition, arc e = ((F, t), (F', t')) is in A, if $t' = \sigma(t)$ and F' derives from F by assigning the demand j to some unique vehicle k. Arc e is associated with a cost W_e, which is defined as the number of additional stops required to assign vehicle k to request j. Whenever $\sigma(t)$ = Nil, we define an arc from (F, t) to End, with cost equal to 0.

Solving SNMP amounts to computing a shortest path in the network G, from the source node (0, (0, 0)) to node End. This is true in any case when $H = 0$. If Q is bounded or fixed, then the size of the State Space F is polynomially bounded in n and m. □

Remark 1. (***Difficult features of SNMP***).

If $H > 0$, then Theorem 2 does not hold since the state set TS should contain, for any *time* (i_0,j_0) of the time space TS, a list of those requests j such that $o(j) = i_0$ modulo n, and which have already been scheduled. So, it follows from Theorems 1 and 2 that, because of the interval structure which is underlying the SNMP, the case when $H = 0$ and loads $L(j)$, $j = 1,\ldots, m$, are all equal to 1, is close to be polynomial in time. From a practical viewpoint, the difficulty will mainly arise from the existence of *waiting laps* and from the load distribution.

3.2. Solving SNMP through the basic ILP model

To examine the practical difficulty of SNMP, we first solved the ILP model described in Section 2.1, under different hypotheses on H and on the loads $L(j)$. More precisely, we analyze the variation of running times and the gap of the solution with respect to the linear relaxation of the model as a function of the main problem parameters, namely n, m and K.

Since no test-bed exists for SNMP, we randomly generate a set of benchmark instances with different sizes. Each instance is denoted by a code *X_n-m-H-L-Q* that summarizes its main characteristics, where X is equal to either N or U for unrestricted and unit load instances, respectively, n is the number of nodes, m the number of requests, H is the maximum number of waiting laps, L is the maximum load of a request and Q is the vehicle capacity. Tests are run using ILOG-CPLEX 12.5 solver on a Linux CentOS release 6.6 server with 4 processors Intel(R) Xeon(R) CPU X5687 @ 3.60 Gigahertz.

Table 1 reports the results we obtained with instances with n = {10, 15} and m= {28, 33, 38} with loads randomly generated in the interval (Bast et al., 2014; Doerner & Salazar-González, 2014) and vehicle capacity Q equal to 20. For each instance we report the corresponding number of n, m, and H. Furthermore, we include:

- K_{min}: minimum number of vehicles;
- K^*: number of vehicles used in the optimal solution
- z^*: optimal solution value of SNMP
- LB: value of the linear relaxation of the model at the root node
- GAP: percentage gap of the linear relaxation with respect to the optimal solution, computed as $(z^* - LB)/z^*$;
- CPU: running time in seconds.

Table 2 reports instead the results for unit loads. In such case, to guarantee that for the smaller values of H the number of required vehicles is greater than 1, we reduced the vehicle capacity Q to 5. The difficulty of the problem seems relatively non sensitive to the number of nodes n while it strongly depends on the number of requests m as shown by Fig. 4 that reports the average solution time for various m values in the test set. Another quite evident issue is that the difficulty increases when the value of H increases: the instances with $H = 0$ are easier than those with $H > 0$. As shown by Fig. 5 the computing time growth with H is considerably steep even if the average percentage gap only moderately worsen between H=0 and 4. Finally, we should observe that in some cases the optimal solution value is obtained with a number of vehicles larger than the minimum possible: this happened in about one third of the 30 instances in the benchmark.

By observing Table 1 we note that model SNMP is not solvable within reasonable computing time as soon as m becomes close to 40, whereas it is relatively less sensible to the value of n. This is not surprising since the quality of the linear relaxation of the ILP model is quite poor showing gaps as large as 50 percent. An immediate explanation of such poor quality of the bound is related to the coupling constraints (4): by relaxing the integrality constraint on Z it is possible to split requests $j = 1,\ldots, m$ into

Table 1
Solution of the ILP model of Section 2.1 for instances with unrestricted loads.

Instance	n	m	H	K_{min}	K^*	z^*	LB	(percent)GAP	CPU
N_10-28-0-10-20	10	28	0	5	6	24	16	33.3	0.91
N_10-28-1-10-20	10	28	1	3	3	21	13	38.1	24.02
N_10-28-2-10-20	10	28	2	2	2	20	12	40.0	73.36
N_10-28-3-10-20	10	28	3	2	2	20	12	40.0	411.53
N_10-28-4-10-20	10	28	4	1	2	20	12	40.0	469.59
N_10-33-0-10-20	10	33	0	5	5	25	15	40.0	3.26
N_10-33-1-10-20	10	33	1	3	3	22	13	40.9	58.94
N_10-33-2-10-20	10	33	2	2	2	21	12	42.9	91.03
N_10-33-3-10-20	10	33	3	2	2	21	12	42.9	2057.87
N_10-33-4-10-20	10	33	4	1	1	21	11	47.6	76.54
N_10-38-0-10-20	10	38	0	5	6	28	16	42.9	176.52
N_10-38-1-10-20	10	38	1	3	3	24	13	45.8	320.27
N_10-38-2-10-20	10	38	2	2	3	23	12	47.8	5663.64
N_10-38-3-10-20	10	38	3	2	2	23	12	47.8	13042.29
N_10-38-4-10-20	10	38	4	1	2	23	12	47.8	70946.74
N_15-28-0-10-20	15	28	0	4	4	24	18	25.0	0.31
N_15-28-1-10-20	15	28	1	2	3	22	17	22.7	1.89
N_15-28-2-10-20	15	28	2	2	2	22	16	27.3	2.86
N_15-28-3-10-20	15	28	3	1	2	22	16	27.3	24.26
N_15-28-4-10-20	15	28	4	1	1	22	16	27.3	9.51
N_15-33-0-10-20	15	33	0	4	4	31	24	22.6	0.78
N_15-33-1-10-20	15	33	1	2	3	28	20	28.6	26.66
N_15-33-2-10-20	15	33	2	2	2	26	18	30.8	44.89
N_15-33-3-10-20	15	33	3	2	2	25	17	32.0	123.18
N_15-33-4-10-20	15	33	4	1	2	25	17	32.0	538.90
N_15-38-0-10-20	15	38	0	6	7	34	23	32.4	42.43
N_15-38-1-10-20	15	38	1	3	4	29	19	34.5	567.92
N_15-38-2-10-20	15	38	2	2	3	28	18	35.7	2637.60
N_15-38-3-10-20	15	38	3	2	2	28	17	39.3	7562.48
N_15-38-4-10-20	15	38	4	2	2	28	17	39.3	48911.54

Table 2
Solution of the ILP model of Section 2.1 for instances with unit loads.

Instance	n	m	H	K_{min}	K^*	z^*	LB	(percent)GAP	CPU
U_10-28-0-1-5	10	28	0	4	4	23	15	34.8	0.85
U_10-28-1-1-5	10	28	1	2	2	20	13	35.0	2.26
U_10-28-2-1-5	10	28	2	2	2	19	12	36.8	12.84
U_10-28-3-1-5	10	28	3	1	1	18	12	33.3	2.01
U_10-28-4-1-5	10	28	4	1	1	18	11	38.9	4.76
U_10-33-0-1-5	10	33	0	4	4	26	18	30.8	1.81
U_10-33-1-1-5	10	33	1	2	2	22	14	36.4	6.33
U_10-33-2-1-5	10	33	2	2	2	21	13	38.1	55.35
U_10-33-3-1-5	10	33	3	1	1	20	12	40.0	20.10
U_10-33-4-1-5	10	33	4	1	1	20	12	40.0	32.54
U_10-38-0-1-5	10	38	0	5	5	25	16	36.0	10.28
U_10-38-1-1-5	10	38	1	3	3	22	13	40.9	86.58
U_10-38-2-1-5	10	38	2	2	2	21	12	42.9	189.21
U_10-38-3-1-5	10	38	3	2	2	21	12	42.9	1525.30
U_10-38-4-1-5	10	38	4	1	1	21	12	42.9	90.18
U_15-28-0-1-5	15	28	0	4	4	23	20	13.0	0.31
U_15-28-1-1-5	15	28	1	2	2	22	18	18.2	0.42
U_15-28-2-1-5	15	28	2	2	2	21	17	19.0	2.59
U_15-28-3-1-5	15	28	3	1	1	21	17	19.0	0.87
U_15-28-4-1-5	15	28	4	1	1	21	16	23.8	1.93
U_15-33-0-1-5	15	33	0	4	4	29	21	27.6	1.02
U_15-33-1-1-5	15	33	1	2	2	25	18	28.0	2.14
U_15-33-2-1-5	15	33	2	2	2	25	17	32.0	41.18
U_15-33-3-1-5	15	33	3	1	1	24	17	29.2	5.80
U_15-33-4-1-5	15	33	4	1	1	24	17	29.2	23.29
U_15-38-0-1-5	15	38	0	5	5	30	21	30.0	4.25
U_15-38-1-1-5	15	38	1	3	3	26	18	30.8	28.87
U_15-38-2-1-5	15	38	2	2	2	25	17	32.0	73.32
U_15-38-3-1-5	15	38	3	2	2	25	17	32.0	247.10
U_15-38-4-1-5	15	38	4	1	1	25	17	32.0	36.51

several small fractional pieces, and distribute them among the vehicles $k = 1,\ldots, K$ and the waiting laps $0,\ldots, H$. It follows that the related values of $T_{i,k}$ may turn out to be very close to 0.

As it could be expected from the results of Section 3.1, the instances with unit load are easier to solve and the gap at the root node is substantially smaller. Moreover, for unit load instances the optimal number of vehicles K^* is always equal to K_{min}.

Because of the disappointing behavior of the continuous relaxation in producing tight lower bounds for the SNMP model we also tried to apply Lagrangean Relaxation. Unfortunately, such

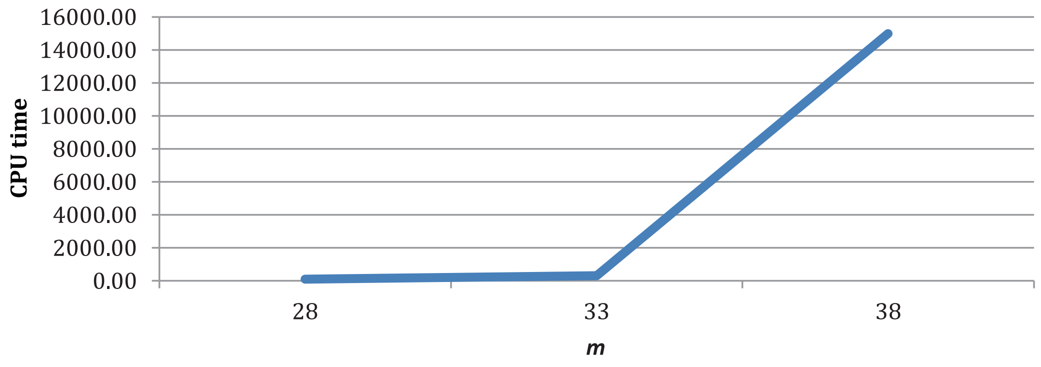

Fig. 4. Average CPU time for the solution of instances with various numbers of requests.

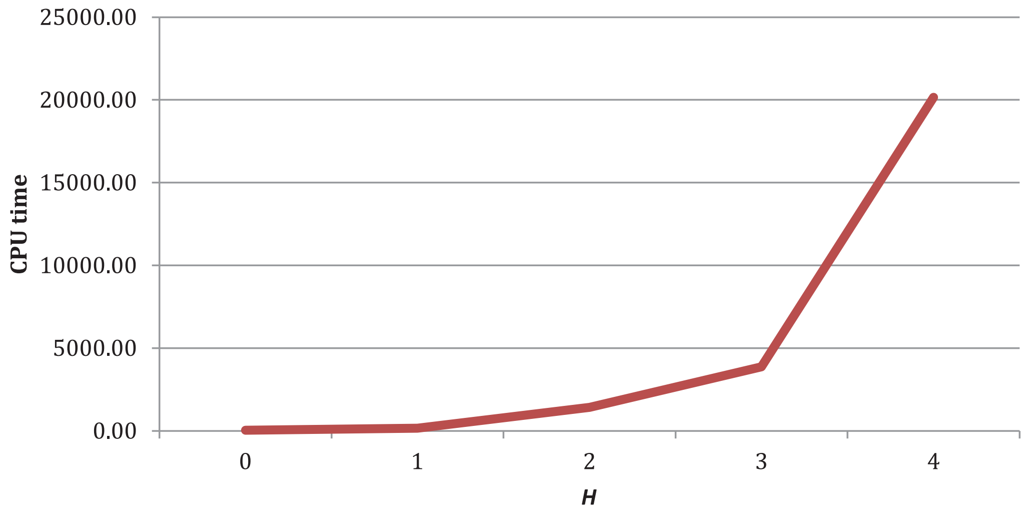

Fig. 5. Average CPU time for the solution of instances with various number of waiting laps.

Table 3
Dependence of the Local Stop Number Stop(h)on the lap index h.

Instance	n	m	H	K*	z*	z*	Stop(1)	Stop(2)	Stop(3)	Stop(4)	Stop(5)
U_10-28-2-1-5	10	28	2	2	4	19	10	7	2	–	–
U_10-28-4-1-5	10	28	4	1	2	18	9	5	3	1	–
U_10-33-2-1-5	10	33	2	2	2	21	11	8	2	–	–
U_10-33-4-1-5	10	33	4	1	1	20	9	6	4	1	–

relaxation did not yield any significant improvement. Some details on the behavior of the Lagrangean Relaxation scheme are given in an Appendix.

3.2.1. Evaluating stop numbers for a specific user

We conducted some specific testing on the number of stops for specific users. The tests show that, if $H = 0$, the average number of times a vehicle k stops while servicing a given user j does not depend on k and only depends on j. More precisely, it is proportional to the ratio $(d(j) - o(j) - 1)/n$ if $d(j) - o(j) > 0$ and to $(n + d(j) - o(j) - 1)/n$ otherwise. However, when $H > 0$ it is interesting to observe how the number of times a vehicle k stops during lap h depends on the value h itself. Such correlation may be summarized in the following Table 3, which considers the instances with unit loads U_10-x-y-1-5, with $x\in\{28, 33\}$, and $y\in\{2, 4\}$. In the table *Stop*(h), which we call the *Local Stop Number* associated with lap h, denotes the number of times an active vehicle stops during lap h.

From the table we can observe that in general, for a given user j, the largest its actual *waiting time* the smallest will be its *travel time*, in term of additional stops between its origin and destination.

3.3. Solving the set-covering based formulation

A better lower bound to SNMP is given by the continuous relaxation of the set-covering based model SC-SNMP described in Section 2.3, which can be solved through column generation.

Let us suppose that we are given an active subset S_0 of the feasible service set S, and that we solved the restriction to S_0 of the SC-SNMP. Then, let $\lambda = (\lambda_j, j = 1,\ldots, m)$ be the dual solution associated with the optimal solution of such restricted problem. Generating a new service $s = (J, h)$ requires solving the following ILP model:

$$\textbf{\textit{Pricing}} \quad \max \sum_{j=1}^{m}\sum_{h=0}^{H}\lambda_j.Z_{jh} - \sum_{i=0}^{p} T_i \tag{18}$$

subject to:

$$\sum_{h=0}^{H} Z_{jh} \le 1, \quad \forall j = 1,\ldots,m \tag{19}$$

$$\sum_{(j,h)\in Cross(i)} L(j)\cdot Z_{jh} \le Q, \quad \forall i = 0, \ \ldots, \ p \tag{20}$$

$$Z_{jh} \le T_i \quad \forall i = 0, \ \ldots, \ p; \ \forall (j,h) \in Stop(i) \tag{21}$$

$$Z_{jh} \in \{0,1\}, \quad \forall j = 1,\ldots,m; \forall h = 0, \ \ldots, \ H \tag{22}$$

$$T_i \cdot \in \{0,1\}, \quad \forall i = 0, \ \ldots, \ p, \tag{23}$$

where the objective function (18) maximizes the cost of the new service, constraints (19) impose that each request is assigned at most once. Constraints (20) are the capacity constraints and (21) are the coupling constraints for the decision variables. Finally, constraints (22) and (23) define the decision variables. Note that the service generated by this ILP model is an *improving column* for the master problem if and only if its cost (18) is strictly positive.

Whenever the above Pricing problem returns a solution that is strictly positive we may add the corresponding feasible service to S_0 and iterate, otherwise the current restricted problem provides the optimal solution to SC-SNMP. The exact solution of SC-SNMP can be obtained through a complete *Branch and Price* algorithm based on such lower bound. However, since our aim is mainly to investigate the quality of the lower bound obtained with such formulation we do not implemented a full Branch and Price algorithm and we restrict our analysis to the root node of the branch-decision-tree.

Because the solution of the pricing problem is extremely time-consuming we adopt a hybrid approach that first tries to construct heuristically services with positive reduced cost computed as in (18) that may be added to the current SC-SNMP and resorts to the exact solution only when heuristics are unsuccessful. For the heuristic step we use several heuristics based on the *Greedy Randomized Adaptive Search Procedure* (GRASP, see (Feo & Resende, 1989) and (Resende & Ribeiro, 2003)), approach followed by a local search step. More precisely, the GRASP is a construction algorithm in which at each iteration the next request to be added to the current service is randomly selected among the R best ones, i.e., those maximizing the associated insertion cost. This, for request j, is computed as the difference between λ_j and the minimum number of additional stops created by the insertion in one of the existing vehicle tours. We also consider a variant in which the insertion is divided by the load of the request so as to favor insertions with largest unit cost. We consider $R\in\{1, 2, 3\}$, where clearly $R = 1$ corresponds to a deterministic greedy. The six resulting GRASP algorithms are followed by a local search procedure that considers two-exchanges between requests. As shown in the following, the GRASP pricing works quite well and provides most of the columns that are used in the calculation of the bound by using a negligible computing time.

According to our experimental testing, by relaxing the integrality constraint of SC-SNMP we obtain very good lower bound values, as shown by the following Table 4 that reports the experiments performed on the same benchmark instances and relative to the root node of a Branch and Price algorithm. For the solution of the pricing problem we use the hybrid approach described above, and we impose an overall time limit of 3 hours of computing time. In Table 5 we instead report the results for unit load instances. The columns of the tables have the same meaning as in the previous tables, while COL_H and COL_E is the total number of generated columns by the heuristic and by the exact pricing algorithms. Note that almost all the CPU time is spent in the exact pricing step of the algorithm as generally happens in similar approaches. The optimal solution values reported in column z^* are those computed with the initial ILP model described in Section 2.1.

While Tables 1 and 2 show a considerable difference between the general and the unit case, this is not the case with the set-covering-based model.

However, the pricing problem turns out to be extremely time consuming, when it is solved as an integer program through CPLEX, making it impossible to incorporate it into an overall complete Branch and Price approach. Furthermore, our experimental analysis shows that relaxing the integrality constraint in the pricing problem provides a very poor lower bound as may be expected due to the very specific nature of the objective function of the pricing problem that is the difference of two terms. Moreover, we may observe that:

Theorem 3. *The pricing problem is NP-Hard, even in the case where $H = 0$, we have unit loads, and $Q = +\infty$.*

Proof. We just need to prove that if W is equal to an integer in $\{1,\ldots, n-1\}$, then solving the pricing problem while imposing that $\Sigma_i T_i = W$ is NP-Hard. Let us consider any non-oriented graph G whose node set is the set $\{1,\ldots, n-1\}$. With every edge (x, y), $x < y$, in this graph we may associate some request j such that $o(j) = x$ and $d(j) = y$, and set $\lambda_j = 1$. Then, computing a solution (Z, T) of *the pricing problem* such that $\Sigma_i T_i = W$ and $\Sigma_j \lambda_j .Z_{j,h} \ge W.(W-1)/2$ requires finding a complete sub-graph of G with exactly W nodes, which is known to be NP-Hard (see, e.g., (Garey & Johnson, 1979)). □

On the other hand, if $H = 0$, and Q is fixed, our problem happens to be polynomial in time:

Theorem 4. *In case Q is fixed and $H = 0$, then the pricing problem can be solved in polynomial time.*

Proof. The pricing problem can be solved through dynamic programming as a shortest path problem defined on an acyclic directed graph F with a number of vertices which depends polynomially by n and m. We define a *state* as being any integer-valued vector $V = (V_1,\ldots,V_Q)$, such that $n \ge V_1 \ge V_2 \ge,\ldots, \ge V_Q \ge -1$. Such *state* corresponds to the possible *load profile functions* of a vehicle when it arrives at node i, while being loaded according to decisions which have been taken at preceding nodes i', i.e. such that $i' \le i$. The vehicle carries a load not smaller than c when arriving at any node i_1 such that $i_1 \le i + V_c$ and such load is going to be less than c when the vehicle leaves node $i + V_c$. The fact that $V_1 = -1$ means that the vehicle is empty when it arrives at i. Clearly, vector V tells us whether the vehicle is supposed to unload at node i, and, whether i is currently scheduled as a *stop node* for the vehicle. We denote by $\mathcal{V}$ the set of all possible states. Then the nodes in the directed graph F are all triples (i, j, V), $i = 0,\ldots, n-1$, $j = 1,\ldots, m$, such that $o(j) = i$, $V \in \mathcal{V}$, augmented with 2 fictitious nodes, called *Start* and *End*. An arc $((i_0, j_0, V_0), (i_1, j_1, V_1))$ exists in F if one of the following cases occurs:

(1) $i_1 = i_0 + 1$, $j_1 = \{\min j: o(j) = i_1\}$, $V_1 = V_0 - 1$. In this case, the vehicle moves from i_0 to $i_0 + 1$ without loading $L(j_0)$;
(2) $i_1 = i_0 + 1$, $j_1 = \{\min j: o(j) = i_1\}$, V_1 derives from V_0 by taking into account the additional load $L(j_0)$ between $o(j_0) = i_0$ and $d(j_0)$ and the fact the vehicle has been moving from i_0 to $i_0 + 1$ after loading $L(j_0)$. In this case, the vehicle moves from i_0 to $i_0 + 1$ after loading $L(j_0)$;
(3) $i_1 = i_0$, $j_1 = \{\min j: o(j) = i_1\}$, $V_1 = V_0$. Here, the vehicle remains at i_0 and does not load $L(j_0)$;
(4) $i_1 = i_0$, $j_1 = \{\min j > j_0,: o(j) = i_1\}$, V_1 derives from V_0 by taking into account the additional load $L(j_0)$ between $o(j_0) = i_0$ and $d(j_0)$. In this case, the vehicle remains at i_0 and loads $L(j_0)$;

In Cases 1 and 3, the arc $((i_0, j_0, V_0), (i_1, j_1, V_1))$ is associated with a null length. In Cases 2 and 4, the arc is associated with

Table 4
Results of the column generation algorithm for instances with unrestricted loads.

.	n	m	H	z*	LB	GAP	COL_H	COL_E	CPU
N_10-28-0-10-20	10	28	0	24	24	0.0	98	7	0.6
N_10-28-1-10-20	10	28	1	21	21	0.0	188	23	9.7
N_10-28-2-10-20	10	28	2	20	20	0.0	241	68	95.6
N_10-28-3-10-20	10	28	3	20	20	0.0	271	92	1014.9
N_10-28-4-10-20	10	28	4	20	20	0.0	273	96	4229.0
N_10-33-0-10-20	10	33	0	25	25	0.0	163	13	1.3
N_10-33-1-10-20	10	33	1	22	21	4.5	299	119	31.8
N_10-33-2-10-20	10	33	2	21	20	4.8	318	186	505.0
N_10-33-3-10-20	10	33	3	21	20	4.8	412	182	3015.0
N_10-33-4-10-20	10	33	4	21	–	–	352	204	T.L.
N_10-38-0-10-20	10	38	0	28	27	3.6	219	69	9.0
N_10-38-1-10-20	10	38	1	24	23	4.2	361	124	81.3
N_10-38-2-10-20	10	38	2	23	23	0.0	431	154	1585.2
N_10-38-3-10-20	10	38	3	23	–	–	445	179	T.L.
N_10-38-4-10-20	10	38	4	23	–	–	426	183	T.L.
N_15-28-0-10-20	15	28	0	24	24	0.0	100	4	0.3
N_15-28-1-10-20	15	28	1	22	22	0.0	170	45	7.1
N_15-28-2-10-20	15	28	2	22	22	0.0	209	65	17.3
N_15-28-3-10-20	15	28	3	22	22	0.0	241	78	61.7
N_15-28-4-10-20	15	28	4	22	22	0.0	201	51	75.9
N_15-33-0-10-20	15	33	0	31	31	0.0	159	10	1.1
N_15-33-1-10-20	15	33	1	28	27	3.6	292	45	15.3
N_15-33-2-10-20	15	33	2	26	25	3.8	363	166	150.3
N_15-33-3-10-20	15	33	3	25	25	0.0	452	216	1001.0
N_15-33-4-10-20	15	33	4	25	25	0.0	469	258	2960.7
N_15-38-0-10-20	15	38	0	34	33	2.9	196	49	2.6
N_15-38-1-10-20	15	38	1	29	29	0.0	398	145	41.2
N_15-38-2-10-20	15	38	2	28	28	0.0	417	214	561.2
N_15-38-3-10-20	15	38	3	28	28	0.0	448	249	5558.1
N_15-38-4-10-20	15	38	4	28	–	–	477	292	T.L.

a length that expresses the impact on the *Stop Number* of the insertion of request j_0 into of the vehicle, taking into account the current state V_0. One easily checks that solving our *Stop Minimization-Price* problem means computing a shortest path in this acyclic digraph *F*. □

Remark 2. (***Theorem 4*** ***and an efficient pricing algorithm***)

In many cases, it happens that a dynamic programming algorithm such as that used to prove Theorem 4, may be adapted to obtain an efficient algorithm, even when time-polynomiality does not apply. Unfortunately, this is not the case here. Clearly, if H is not 0, the main assumption of our algorithmic scheme is not relevant any more. However, even when $H = 0$, the number of non-dominated states in V tends to become large very fast. Indeed, we implement the above algorithmic scheme by linearly ordering the request set $D = \{1..m\}$ in a way which is consistent with the natural ordering on the *origin* set $\{o(j), j = 1..m\}$, and by considering that, once some request *j* has been inserted into the service *J*, the decision is about the next request *j*' in *D* which is going to be in *J*. Then the state space V is as in the proof of Theorem 4, and corresponds to all possible *load profile functions* which may characterize the state of the vehicle after *j* has been loaded. State set V may be filtered by a *dominance* relation, but, because of the complex structure of the elements of V, such relation lacks the strength to filter V in an efficient way. As a consequence, the computing time quickly explode as illustrated in the following Table 6. The table reports the time, in seconds, required to solve the pricing problem with either CPLEX 12 or with the dynamic program (DP) for some instances with unrestricted and unit loads and various values of *n*, *m* and *Q*.

Table 6 clearly shows that, unfortunately, the DP approach lacks the efficiency that would be necessary to obtain tight lower bounds for large instances.

4. A hybrid GRASP-R&R heuristic for SNMP

In practical applications the *Stop Number Minimization Problem* must be solved quickly to be used in the real-time management of autonomous vehicles. Clearly the exact methods proposed in the previous sections are not suitable in such a context due to the very long time they require to determine the solutions. Therefore, in this section we propose a fast heuristic approach for SNMP. Although we present a complete method that works on the static problem, so that we can compare its results with those of the exact methods, the approach is designed to be used also in the *on-line* context, so as to define the routing of the vehicles as soon as new requests are revealed. This possibility is guaranteed by the structure of the algorithm that is based on a simple insertion mechanism. At a given time instant of the planning horizon some vehicles are already partially loaded with requests that were revealed before and the currently arriving one is inserted in the best position of the current tours.

The overall algorithm we propose is a hybrid one, which combines sequentially a GRASP (see (Feo & Resende, 1989) and (Resende & Ribeiro, 2003)), to quickly construct an initial solution, followed by a fast refining procedure based on the *Ruin-and-Recreate* (R&R, see (Schrimpf, Schneider, Stamm-Wilbrandt, & Dueck, 2000)) paradigm. The two phases are repeated for a prefixed number ρ of replications and the best solution found over all replications is returned. In the following we describe in detail the various components of the algorithm.

4.1. The GRASP step

Since finding a feasible solution using exactly *K* vehicles is not guaranteed for a heuristic method, the algorithm accepts in the initialization phase to construct solutions having more than *K* vehicles. The number of used vehicles is possibly reduced in the R&R step.

Table 5
Results of the column generation algorithm for instances with unit loads.

Instance	n	m	H	z^*	LB	GAP	COL_H	COL_E	CPU
U_10-28-0-1-5	10	28	0	23	22	4.3	174	12	1.3
U_10-28-1-1-5	10	28	1	20	19	5.0	314	61	10.1
U_10-28-2-1-5	10	28	2	19	18	5.3	354	141	41.7
U_10-28-3-1-5	10	28	3	18	18	0.0	393	172	181.0
U_10-28-4-1-5	10	28	4	18	18	0.0	391	189	834.1
U_10-33-0-1-5	10	33	0	26	26	0.0	207	15	1.6
U_10-33-1-1-5	10	33	1	22	21	4.5	332	111	27.2
U_10-33-2-1-5	10	33	2	21	20	4.8	451	177	249.9
U_10-33-3-1-5	10	33	3	20	20	0.0	531	259	980.4
U_10-33-4-1-5	10	33	4	20	20	0.0	486	256	3796.5
U_10-38-0-1-5	10	38	0	25	25	0.0	243	34	1.9
U_10-38-1-1-5	10	38	1	22	21	4.5	450	130	24.3
U_10-38-2-1-5	10	38	2	21	20	4.8	529	279	562.4
U_10-38-3-1-5	10	38	3	21	20	4.8	533	269	4041.0
U_10-38-4-1-5	10	38	4	21	–	–	490	271	T.L.
U_15-28-0-1-5	15	28	0	23	23	0.0	202	18	0.5
U_15-28-1-1-5	15	28	1	22	21	4.5	224	55	10.3
U_15-28-2-1-5	15	28	2	21	21	0.0	289	113	42.0
U_15-28-3-1-5	15	28	3	21	21	0.0	269	157	97.1
U_15-28-4-1-5	15	28	4	21	21	0.0	275	116	146.8
U_15-33-0-1-5	15	33	0	29	29	0.0	198	27	2.5
U_15-33-1-1-5	15	33	1	25	25	0.0	366	115	22.1
U_15-33-2-1-5	15	33	2	25	24	4.0	420	191	204.7
U_15-33-3-1-5	15	33	3	24	24	0.0	472	278	820.9
U_15-33-4-1-5	15	33	4	24	24	0.0	572	280	2873.2
U_15-38-0-1-5	15	38	0	30	30	0.0	241	26	3.9
U_15-38-1-1-5	15	38	1	26	25	3.8	476	176	30.7
U_15-38-2-1-5	15	38	2	25	25	0.0	474	283	244.2
U_15-38-3-1-5	15	38	3	25	24	4.0	602	347	1311.6
U_15-38-4-1-5	15	38	4	25	24	4.0	558	385	5273.3

Table 6
Using CPLEX and Dynamic Programming (DP) to perform the column generation step.

Request-Type	n	m	Q	CPLEX(s)	DP (s)
unrestricted	10	28	5	8	35
unrestricted	15	28	5	6	52
unit	10	28	5	8	699
unit	10	33	5	15	>3600

The GRASP-based algorithm starts from an empty solution, and then iteratively selects the next not yet serviced request according to a *priority rule* that takes into account the current loading status of the vehicles in the interval between the origin and destination of the request. More precisely, for each not yet serviced request j we compute a priority score P_j, defined as the number of feasible insertion points in the active vehicles which have sufficient residual capacity to carry its load $L(j)$ between the origin node $o(j)$ and the destination node $d(j)$ for some waiting lap $h \leq H$. Whenever the request has at least one feasible insertion in a vehicle that share the stop with an already inserted one, with probability 0.5 the priority score is further reduced by 1. If no feasible insertion exists for request j in the active vehicles we conventionally set $P_j = -1$.

At the current iteration, the next request j' is randomly selected among those minimizing the value P_j over all not yet serviced ones. Note that, if $P_{j'} = -1$, i.e., no feasible insertion point exists for request j' in one of the active vehicles, then a new vehicle is initialized with request j' and a new iteration is started.

Once the next request j' (with $P_{j'} \geq 0$) is selected, it is inserted in the solution as follows. We first examine all feasible insertion points in the active vehicles and, for each such point, we consider the number of additional stops resulting from the insertion. Let σ be the smallest number of additional stops found and let $T(j',\sigma)$ be the set of feasible insertion points for j' associated with value σ. More precisely, $T(j',\sigma) = \{(k,h)$: j' can be inserted in vehicle k with h waiting laps and introducing σ additional stops$\}$. We randomly select the insertion point among those in set $T(j',\sigma)$.

During our extensive preliminary computational testing, we examined different strategies to implement both the selection and the insertion criteria in the above algorithm. For example, we consider in the insertion step also the residual capacity of the vehicle around the insertion point to favor insertions in less filled areas. However, the algorithm we just described here is the one that experimentally provides the best compromise between solutions quality and required running times.

Furthermore, the testing shows that performing an increasing number of replications is beneficial on the algorithm performance given that the computing time needed for a given iteration is very small for the instances of our benchmark. The marginal benefit of incrementing the number of iterations is reported in Fig. 6. Such figure shows the evolution of the average gap of the GRASP solution with respect to the optimal one, when ρ is increased from 1 to 100,000. The figure also shows the considerable benefit obtained by combining the simple and fast GRASP algorithm with the R&R step.

In our computational testing we used 10,000 iterations that represent a good compromise between computing time and quality.

4.2. *The* Ruin-and-Recreate *step*

Following the *Ruin-and-Recreate* idea, the refinement step possibly improves the solution obtained by the GRASP by alternating two phases. In the Ruin phase the current solution is partially destroyed by removing from it some of the requests. Then the solution is reconstructed in the Recreate step by reinserting the removed requests using again the GRASP algorithm.

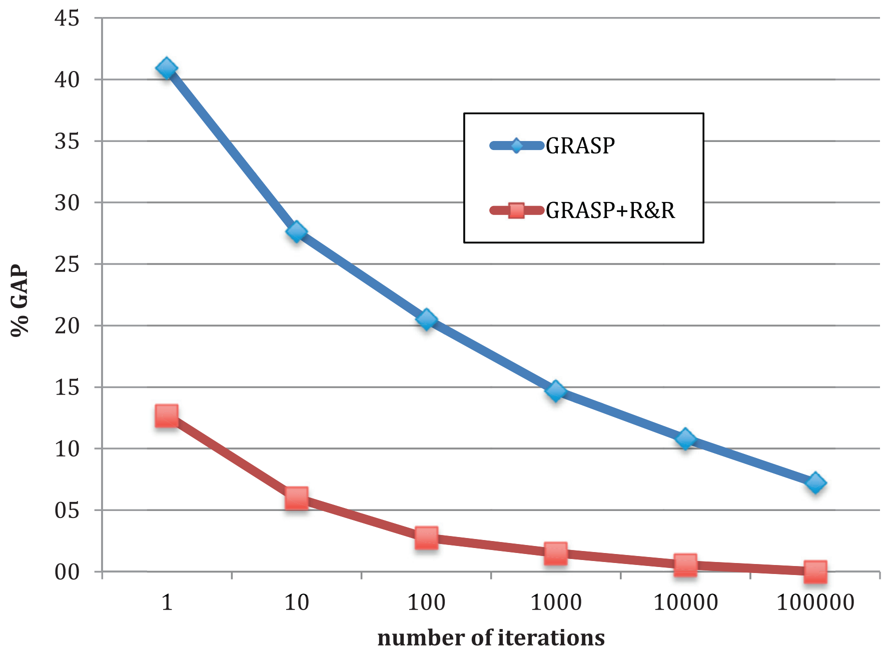

Fig. 6. Evolution of the percentage gap of the solution of the GRASP and GRASP+R&R as a function of the number of iterations.

Table 7
Results of the heuristic approaches on instances with unrestricted loads.

Instance	n	m	H	z*	GRASP			GRASP+R&R-Fast			GRASP+R&R		
					UB	GAP	CPU	UB	GAP	CPU	UB	GAP	CPU
N_10-28-0-10-20	10	28	0	24	26	8.3	0.3	24	0.0	0.8	24	0.0	7.5
N_10-28-1-10-20	10	28	1	21	23	9.5	0.4	21	0.0	0.7	21	0.0	4.0
N_10-28-2-10-20	10	28	2	20	21	5.0	0.4	21	5.0	0.7	20	0.0	3.6
N_10-28-3-10-20	10	28	3	20	22	10.0	0.5	20	0.0	0.8	20	0.0	4.1
N_10-28-4-10-20	10	28	4	20	21	5.0	0.5	21	5.0	0.9	20	0.0	4.6
N_10-33-0-10-20	10	33	0	25	26	4.0	0.4	25	0.0	0.6	25	0.0	3.4
N_10-33-1-10-20	10	33	1	22	24	9.1	0.5	23	4.5	0.7	22	0.0	3.2
N_10-33-2-10-20	10	33	2	21	24	14.3	0.6	22	4.8	0.7	21	0.0	3.0
N_10-33-3-10-20	10	33	3	21	23	9.5	0.6	22	4.8	0.8	21	0.0	3.6
N_10-33-4-10-20	10	33	4	21	22	4.8	0.6	22	4.8	0.7	21	0.0	3.0
N_10-38-0-10-20	10	38	0	28	30	7.1	0.5	28	0.0	0.8	28	0.0	9.5
N_10-38-1-10-20	10	38	1	24	28	16.7	0.6	26	8.3	0.9	24	0.0	4.6
N_10-38-2-10-20	10	38	2	23	26	13.0	0.7	24	4.3	0.9	24	4.3	4.5
N_10-38-3-10-20	10	38	3	23	27	17.4	0.8	25	8.7	1.1	24	4.3	5.0
N_10-38-4-10-20	10	38	4	23	26	13.0	0.8	25	8.7	1.1	24	4.3	5.6
N_15-28-0-10-20	15	28	0	24	24	0.0	0.3	24	0.0	0.6	24	0.0	4.0
N_15-28-1-10-20	15	28	1	22	24	9.1	0.4	22	0.0	0.7	22	0.0	3.9
N_15-28-2-10-20	15	28	2	22	24	9.1	0.5	22	0.0	0.8	22	0.0	4.0
N_15-28-3-10-20	15	28	3	22	24	9.1	0.5	22	0.0	0.7	22	0.0	4.2
N_15-28-4-10-20	15	28	4	22	23	4.5	0.5	22	0.0	0.8	22	0.0	3.7
N_15-33-0-10-20	15	33	0	31	33	6.5	0.4	31	0.0	0.9	31	0.0	16.1
N_15-33-1-10-20	15	33	1	28	31	10.7	0.5	28	0.0	1.1	28	0.0	8.9
N_15-33-2-10-20	15	33	2	26	29	11.5	0.6	26	0.0	1.1	26	0.0	7.7
N_15-33-3-10-20	15	33	3	25	27	8.0	0.7	26	4.0	1.3	25	0.0	8.1
N_15-33-4-10-20	15	33	4	25	28	12.0	0.7	26	4.0	1.2	25	0.0	8.9
N_15-38-0-10-20	15	38	0	34	38	11.8	0.6	34	0.0	1.4	34	0.0	14.6
N_15-38-1-10-20	15	38	1	29	36	24.1	0.8	31	6.9	1.5	30	3.4	10.2
N_15-38-2-10-20	15	38	2	28	35	25.0	0.8	30	7.1	1.6	28	0.0	10.1
N_15-38-3-10-20	15	38	3	28	32	14.3	1.0	30	7.1	1.7	28	0.0	9.3
N_15-38-4-10-20	15	38	4	28	34	21.4	1.1	29	3.6	1.9	28	0.0	10.5
Average				24.3	27.0	10.8	0.6	25.1	3.1	1.0	24.5	0.5	6.4

The key factor of the R&R algorithm is clearly the Ruin mechanism, for which several possible strategies were proposed in the literature (see, e.g. (Schrimpf et al., 2000)). Those strategies are often combined together by introducing a random sorting decision mechanism that allows switching from a strategy to another one as in the Adaptive Large Neighborhood Search (see (Pisinger & Ropke, 2007)). In our implementation we want to keep a high speed and considerable simplicity, hence we adopt two simple Ruin mechanisms, thus getting two different implementations of the whole R&R algorithm.

Let us call unpaired requests all requests that do not share both stops at origin or destination with other requests. The first Ruin mechanism, R1, is a deterministic one that concentrates on the unpaired requests and removes all of them from the current solu-

Table 8
Sensitivity Analysis of the heuristic performance out of 10 runs.

Instance	n	m	H	z*	GRASP (10 runs)			GRASP+R&R (10 runs)		
					best	worst	average	best	worst	average
N_10-28-0-10-20	10	28	0	24	31	36	33.4	24	26	24.9
N_10-28-1-10-20	10	28	1	21	25	35	30.6	22	24	23.2
N_10-28-2-10-20	10	28	2	20	25	33	29	22	27	23.5
N_10-28-3-10-20	10	28	3	20	25	33	29.9	22	27	23.3
N_10-28-4-10-20	10	28	4	20	23	34	28	21	25	22.6
N_10-33-0-10-20	10	33	0	25	29	33	30.8	26	28	27.2
N_10-33-1-10-20	10	33	1	22	28	33	30.3	23	27	24.9
N_10-33-2-10-20	10	33	2	21	28	33	30.7	22	28	25.1
N_10-33-3-10-20	10	33	3	21	26	34	29.8	24	26	24.8
N_10-33-4-10-20	10	33	4	21	24	31	27.8	23	26	24.6
N_10-38-0-10-20	10	38	0	28	36	41	38.2	28	32	30
N_10-38-1-10-20	10	38	1	24	32	35	33.7	28	33	29.1
N_10-38-2-10-20	10	38	2	23	31	37	34.1	25	30	27.6
N_10-38-3-10-20	10	38	3	23	31	37	33.2	25	28	26.9
N_10-38-4-10-20	10	38	4	23	31	36	33.7	26	29	27.5
N_15-28-0-10-20	15	28	0	24	28	34	30.1	24	26	24.8
N_15-28-1-10-20	15	28	1	22	28	35	30.4	23	28	24.5
N_15-28-2-10-20	15	28	2	22	28	32	29.5	22	28	23.8
N_15-28-3-10-20	15	28	3	22	26	33	30.1	23	26	24.2
N_15-28-4-10-20	15	28	4	22	26	33	29	23	26	24
N_15-33-0-10-20	15	33	0	31	37	44	39.9	31	35	32.3
N_15-33-1-10-20	15	33	1	28	35	43	39.9	28	31	29.3
N_15-33-2-10-20	15	33	2	26	35	43	37.8	27	30	28.3
N_15-33-3-10-20	15	33	3	25	34	40	37.6	28	31	29.5
N_15-33-4-10-20	15	33	4	25	32	40	36.1	26	32	28.7
N_15-38-0-10-20	15	38	0	34	40	50	46	34	36	35.1
N_15-38-1-10-20	15	38	1	29	41	50	45.4	31	36	32.6
N_15-38-2-10-20	15	38	2	28	40	47	42.9	30	36	32.9
N_15-38-3-10-20	15	38	3	28	41	46	43	30	33	31.7
N_15-38-4-10-20	15	38	4	28	38	46	42.8	31	36	32.8
Average				24.3	31.1	37.9	34.5	25.7	29.5	27.3

tion. The second mechanism is instead a mixed one that combines R1, chosen with probability π, with a probabilistic one, called R2 and chosen with probability $1-\pi$, where each request $j = 1, \ldots, m$ may be removed with probability π'. During our extensive preliminary testing we attempt several other Ruin rules but the two which we just described represent the best performing ones. In particular, rule R1 is particularly well suited for a fast execution in which just a few iterations are performed, whereas the probabilistic rule R2 works best when more time is available for the iterations.

The R&R algorithm has been implemented in two versions. The first one, called R&R-Fast is a simple descent algorithm that uses Ruin mechanism R1 described above. As soon as a non-improving solution is found the algorithm is stopped and the current best solution is returned. Clearly such a method is suited to quickly polish the solutions obtained with the GRASP and is sufficiently well performing to be used within on-line contexts. A better version, simply called R&R hereafter, is a more traditional implementation where the search is continued even in case a non-improving solution is found. More precisely, ruin mechanism R2 is used, combined with a probabilistic acceptance rule (see also (Pisinger & Ropke, 2007)) where the current non-improving solution is accepted with probability $\max\{0,(0.9 - \Delta/10)\}$ where Δ is the number of additional stops with respect to the current best solution. The process is iterated until no improvement is found for γ consecutive R&R iterations.

4.3. Testing of the hybrid algorithm

To have a clear assessment on the performance of the proposed algorithm we test it on the instances with unrestricted load that we previously solved with the ILP exact approach. An intensive preliminary testing allows us to define the parameters set that offers the best compromise between quality and computing time, which is: ρ =10,000, $\pi = 0.75$, $\pi' = 5$ and $\gamma = 10$.

As in the case of the experiments which we performed on the ILP formulation, we ran the tests while using a Linux CentOS release 6.6 server with 4 processors Intel(R) Xeon(R) CPU X5687 @ 3.60 Gigahertz, together with a C++ gcc version 4.4.7 compiler.

The results are summarized in Tables 7 and 8. For each instance Table 7 reports the optimal solution value z*. Then, we give the results of the simple GRASP, i.e., without the R&R step, for the GRASP followed by the simple descent R&R-Fast and the GRASP followed by the complete R&R. For each algorithm we give the value of the solution, the percentage gap with respect to the optimal one and the computing time in seconds. In Table 8, we test the sensitivity of the results produced by our GRASP algorithms to their random control components, by providing, for the same instances and a number of runs equal to 10, the average, best and worse value produced by both simple GRASP and GRASP+R&R.

Table 7 shows that the proposed algorithm is very effective. In fact the complete GRASP+R&R finds, in a few seconds of computing time, 26 out of 30 optimal solutions and in the remaining cases uses at most one additional stop. The computing times are also growing slowly with the value of m, making our algorithm suitable also for larger instances involving several tens of stops. Moreover, in cases where faster running times are needed (e.g. in the on-line context) the more aggressive GRASP+R&R-Fast is still able to find 50 percent of optimal solutions in about one second. Finally, we should observe that the R&R step is really worth being used since it greatly improves the GRASP solutions with a modest computing time requirement.

Table 8 shows that two distinct runs may produce results with very different quality levels when we restrict ourselves to the use of the simple greedy GRASP scheme, and that this strong dependence to random sorting is drastically reduced in the case of GRASP+R&R.

5. Conclusions

In this paper we considered a problem arising in the monitoring of a fleet of autonomous electric vehicles for which we defined the scheduling that minimizes the overall number of stops performed to serve a set of pickup and delivery requests. The problem turned out to be particularly difficult to solve exactly: we proposed two different models and used them to derive bounds and determine the optimal solution for instances with up to 38 requests. We also developed a fast and effective hybrid heuristic that is also suitable for the solution of on-line or semi-online versions of the problem.

To the best of our knowledge this problem was never studied before in the literature, therefore our work represent a starting point for further research, particularly with respect to exact methods. Clearly more sophisticated optimization approaches based on the Branch-and-Price scheme can attempt the solution of larger instances providing also tighter bounds.

Another direction for further research lies on the possibility of considering different network topologies such as trees or more complex and connected ones that may introduce the need of explicitly considering routing as part of the problem. Also additional problem characteristics, such as time windows, or the battery charging times, may add interesting components to the problem that are relevant in practical applications.

Finally, the actual inclusion of algorithms in the *on-line* context may require the evaluation of algorithms with better or guaranteed performance in real time.

Appendix. A Lagrangean lower bound for the SNMP model

A possible way of obtaining better bounds for the SNMP ILP model with respect to the continuous relaxation is represented by defining a Lagrangean relaxation of the coupling constraints (4), which relates vectors Z and T. The resulting Lagrangean function $Lag(Z, T, \lambda)$ is:

$$\begin{aligned} Lag(Z,T,\lambda) &= \sum_{k=1}^{K}\sum_{i=0}^{n} T_{k,i} - \sum_{k=1}^{K}\sum_{i=0}^{n}\sum_{h=0}^{H} \lambda_{i,k,h}\left(T_{k,i} - Z_{i,k,h}\right) \\ &= \sum_{k=1}^{K}\sum_{i=0}^{n} T_{k,i}\left(1 - \sum_{h=0}^{H}\lambda_{i,k,h}\right) + \sum_{k=1}^{K}\sum_{i=0}^{n}\sum_{h=0}^{H}\lambda_{i,k,h}Z_{i,k,h}. \end{aligned}$$

The minimization of $Lag(Z, T, \lambda)$ can be easily obtained through separation. The restriction of the sub-problem to vector T is trivial. The restriction of the sub-problem to vector Z instead requires to compute $Z = (Z_{j,k,h}, j = 1,\dots, m, k = 1,\dots, K, h = 0,\dots, H)$ with $\{0, 1\}$ values, which is a feasible solution of LV(K), and which minimizes the quantity $\sum_{k=1}^{K}\sum_{i=0}^{n}\sum_{h=0}^{H}\lambda_{i,k,h}Z_{i,k,h}$. As discussed in Section 3.1, such a problem can be handled rather efficiently by CPLEX, especially in the case when $H = 0$, that means when it appears as an extension of the interval graph coloring problem.

However the lower bound which results from such Lagrangean relaxation proved experimentally to be not satisfactory, and hardly improves the lower bound which derives from the continuous relaxation. This may be explained by the fact that the integrality constraint has no impact on the optimal value of the above sub-problem on T; as for the sub-problem on Z, the error induced by relaxing the integrality constraint on Z is not very high (usually not larger than 15 percent or 20 percent). As a consequence, the gap between $\text{Inf}_{Z,\ T\ \text{integral}}\ Lag(Z, T, \lambda)$ and $\text{Inf}_{Z,\ T\ \text{rational}}\ Lag(Z, T, \lambda)$ usually does not exceed 10 percent. But $\text{Sup}_{\lambda}\ \text{Inf}_{Z,\ T\ \text{rational}}\ Lag(Z, T, \lambda)$ is exactly equal (by Duality Theory) to the optimal value of the continuous relaxation of SNMP. It follows that, in most cases, the value $\text{Sup}_{\lambda}\ \text{Inf}_{Z,\ T\ \text{integral}}\ Lag(Z, T, \lambda)$ does not improve this value by more than 5 percent to 10 percent, and so does not provides us with a significantly better lower bound than the standard one.

References

Bast, H., Delling, D., Goldberg, A., Muller-Hannemann, M., Pajor, T., Sanders, P., et al. (2014). Route planning in transportation network. Technical Report MSR-TR 2014-4-8, Microsoft Research

Boerndorfer, R., Groetschel, M., Klostermeiner, F., & Kuttner, C. (1999). Telebus Berlin: Vehicle routing scheduling in a dial a ride system. In N. Wilson (Ed.), Computer-Aided Transit Scheduling, *vol. 471 Lecture Notes in Economics and Mathematical Systems* (pp. 391–422). Berlin Heidelberg: Springer.

Boyaci, B., & Zografos, K. (2015). An optimization framework for the development of efficient one way car-sharing systems. *European Journal of Operational Research, 2*, 718–733.

Cordeau, J-F., & Laporte, G. (2007). Dial-a-Ride: models and algorithms. *Annals of Operations Research, 153*, 29–46.

Cordeau, J-F. (2006). A branch and cut algorithm for the Dial-a-Ride. *Operations Research, 54*, 573–586.

Cordeau, J-F., Gendreau, M., Laporte, G., Potvin, J. Y., & Semet, F. (2002). A guide to vehicle routing heuristics. *Journal of the Operational Research Society, 53*, 512–522.

Cornaz, D., & Jost, V. (2008). A one to one correspondence between coloring and stable sets. *Operations Research Letters, 36*, 673–676.

Doerner, K. F., & Salazar-González, J-J. (2014). Pickup-and-Delivery Problems for People Transportation. In P. Toth, & D. Vigo (Eds.), *Vehicle Routing: Problems, Methods and Applications, Ch. 6* (pp. 193–212). Philadephia, MD: SIAM.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters, 8*, 67–71.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY: W. H. Freeman & Co.

Ghiani, G., Lagana, D., Laporte, G., & Mari, F. (2010). Ant colony optimization for the arc routing problem with intermediate facilities under capacity and length restrictions. *Journal of Heuristics, 16*(2), 211–233.

Golumbic, M. C. (2004). *Algorithmic Graph Theory and Perfect Graphs* (2nd edition). Amsterdam: Elsevier.

Luo, Y., & Schonfeld, P. (2007). Reinsertion heuristic for static Dial-a-Ride. *Transportation Research Part B, 41*, 736–755.

Malaguti, E., Monaci, M., & Toth, P. (2011). An exact approach for the vertex coloring problem. *Discrete Optimization, 8*, 174–190.

Mehrotra, A., & Trick, M. J. (1996). A column generation approach for graph coloring. *INFORMS Journal on Computing, 8*, 344–354.

Quilliot, A., & Deleplanque, S. (2015). Constraint propagation for the Dial-a-Ride problem with split loads. In S. Fidanovka (Ed.), *Studies in Computational Intelligence: 470. Recent Advances in Computational Optimization* (pp. 31–50). Switzerland: Springer International Publishing.

Resende, M. G. C., & Ribeiro, C. C. (2003). Greedy randomized adaptive search procedures. In F. Glover, & G. Kochenberger (Eds.), *Handbook of Metaheuristics* (pp. 219–249). Norwell, MA: Kluwer Academic Publishers.

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research, 34*, 2403–2435.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record breaking optimization results-using the ruin & recreate principle. *Journal of Computational Physics, 159*, 139–171.